Software-Projekt

Prof. Dr. Rainer Koschke

Fachbereich Mathematik und Informatik
Arbeitsgruppe Softwaretechnik
Universität Bremen

Wintersemester 2008/09

Überblick I

Anforderungsanalyse

Lernziele

- Ist- und Soll-Zustand ermitteln können
- Anforderungsspezifikation schreiben können
- Anforderungsspezifikation begutachten können

Lernziele

- Ist- und Soll-Zustand ermitteln können
- Anforderungsspezifikation schreiben können
- Anforderungsspezifikation begutachten können

Bevor ein System implementiert werden kann, müssen die Anforderungen an das System erhoben werden. Dieser Aufgabe widmet sich die Anforderungsanalyse. Ihr Ziel ist es, einen möglichst genauen und vollständigen Katalog von Anforderungen an ein System zu erfassen und zu beschreiben. Die Anforderungen beschreiben ein Soll-Konzept, das Verbesserung für ein Problem des Kunden schaffen soll. Damit dieses Konzept auch wirklich hilft, ist es notwendig, zuerst einmal zu verstehen, was das Problem des Kunden ist. Insofern ist die Analyse des Ist-Zustands und seiner Probleme Teil einer Anforderungsanalyse.

Das Ergebnis der Anforderungsanalyse ist der Katalog der Anforderungen. In welcher Form dieser Katalog beschrieben ist, hängt vom Vorgehensmodell ab. Bei dokumentengetriebenen Vorgehensmodellen – wie dem Wasserfallmodell – werden die Anforderungen in Form einer Anforderungsspezifikation schriftlich festgehalten. In vielen agilen Vorgehensmodellen erfolgt die Anforderungsspezifikation als Zuruf des Kunden.

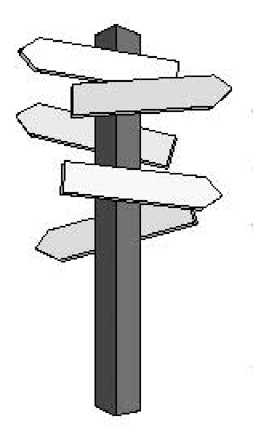
Wann und wie oft die Anforderungsanalyse durchgeführt wird, hängt ebenso stark vom Vorgehensmodell zur Softwareentwicklung ab. Beim Wasserfallmodell wird die Anforderungsanalyse nur einmal und ganz zu Anfang durchgeführt. Beim inkrementellen Vorgehen, bei dem die Entwicklung in kleinen abgeschlossenen Inkrementen des zu entwickelnden Systems erfolgt, wird sie jedes Mal wieder vor der Entwicklung jedes Inkrements durchgeführt. Beim Extreme-Programming hat man durch die Präsenz eines Kundenvertreters vor Ort eine fast kontinuierlich stattfindende Anforderungsanalyse.

In diesem Modul geht es um die Durchführung der Anforderungsanalyse. Wir streben hier die schriftliche Dokumentation der Anforderungen in Form einer Anforderungsspezifikation an, da dies in den meisten Fällen eine Reihe von Problemen vermeidet. Der Inhalt dieses Abschnitts ist jedoch auch für agiles Vorgehen ohne schriftliche Spezifikation der Anforderungen relevant, weil auch orale Überlieferung zu allen Aspekten der Anforderungsspezifikation Stellung nehmen muss.

Übersicht:

Zunächst widmen wir uns der Frage, warum die Anforderungsanalyse so wichtig aber leider auch schwierig ist. Dann lernen wir die grundsätzlichen Schritte der Anforderungsanalyse kennen. Dazu gehören die Erfassung des Ist-Zustands und seiner Probleme durch die Ist-Analyse, die angestrebte Verbesserung durch ein Soll-Konzept durch die Soll-Analyse und die genaue und vollständige Beschreibung der Anforderungen in Form einer Anforderungsspezifikation. Wir vertiefen alle Schritte und lernen Techniken für diese einzelnen Schritte kennen.

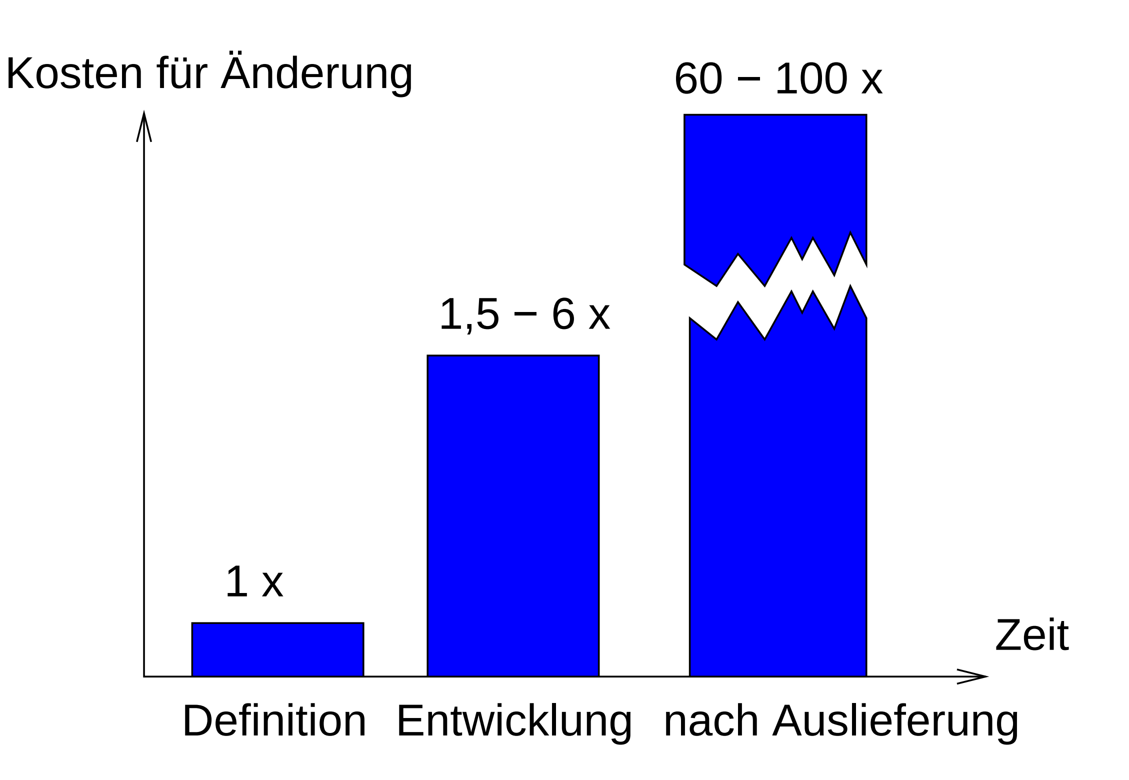
Wegweiser



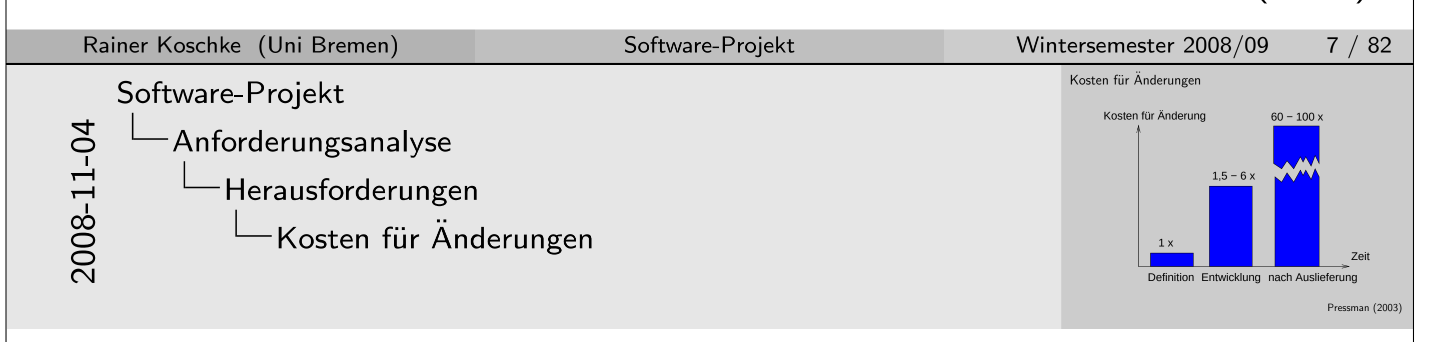
Herausforderungen

Warum sind die Anforderungen so kritisch?
Warum ist ihre Erhebung so schwer?

Kosten für Änderungen



Pressman (2003)

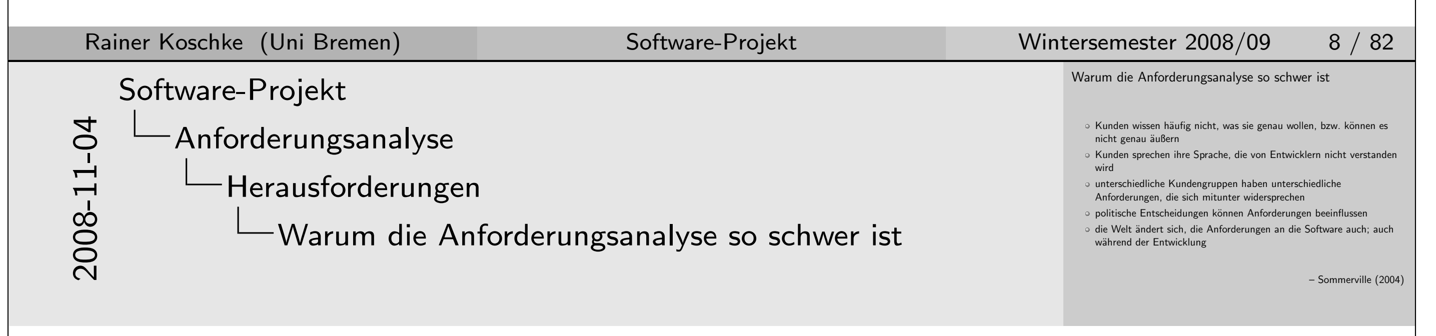


Die Anforderunganalyse ist von zentraler Bedeutung für den Erfolg eines Projektes. Passieren hier Fehler, entsteht ein System, was an den Bedürfnissen des Kunden vorbei entwickelt und damit unbrauchbar wird. Fehler in der Anforderungsanalyse können zwar oft noch in späteren Phasen entdeckt und dann korrigiert werden, die Korrektur dieser Fehler ist jedoch meist mit einem erheblichen Mehraufwand verbunden. Pressman (2003) berichtet von den relativen Kosten für Korrekturen von Fehlern in der Anforderungsanalyse in Abhängigkeit vom Zeitpunkt, zu dem diese Fehler bekannt werden. Findet man einen Fehler der Anforderungsspezifikation in späteren Aktivitäten der Entwicklung, also beim Entwurf, der Implementierung oder dem Test noch vor Auslieferung, dann können die Kosten um einen Faktor 1,5 bis 6 höher sein als die Korrekturkosten, die entstanden wären, hätte man den Fehler noch während der Anforderungsanalyse gefunden. Wird der Fehler noch später gefunden, also nach der Auslieferung der Software, kann er gar um einen Faktor 60 bis 100 höher sein. Es entstehen dann nicht nur zusätzliche Kosten für die Nachbesserung von Anforderungsspezifikation und Entwurf, Fehlerkorrektur und Restrukturierung der Implementierung sowie Wiederholung des Tests. Es entstehen darüber hinaus auch Kosten für die Beseitigung von Auswirkungen des Fehlers. Hat eine eingebettete Software in einem Automobil beispielsweise einen Fehler, so müssen alle Fahrzeuge in der Werkstatt überholt werden. Dadurch entstehen hohe direkte Kosten, aber auch noch indirekte Kosten durch den Imageverlust des Automobilherstellers, wenn sich der Imageverlust auf die Verkaufszahlen auswirkt.

Warum die Anforderungsanalyse so schwer ist

- Kunden wissen häufig nicht, was sie genau wollen, bzw. können es nicht genau äußern
- Kunden sprechen ihre Sprache, die von Entwicklern nicht verstanden wird
- unterschiedliche Kundengruppen haben unterschiedliche Anforderungen, die sich mitunter widersprechen
- politische Entscheidungen können Anforderungen beeinflussen
- die Welt ändert sich, die Anforderungen an die Software auch; auch während der Entwicklung

– Sommerville (2004)

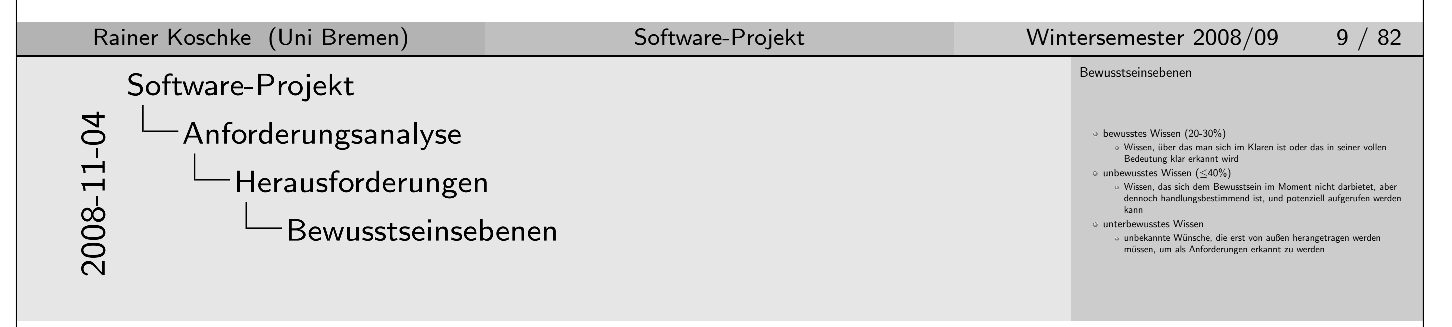


Die Anforderungsanalyse ist schwierig, so dass die Wahrscheinlichkeit, dass wir Fehler machen, recht hoch ist. Sommerville (2004) nennt hierfür eine Reihe von Gründen:

- Die Kunden wissen häufig nicht genau, was sie genau wollen, beziehungsweise können es nicht genau äußern.
- Kunden sind Experten in ihrem Anwendungsbereich, haben jedoch in der Regel wenig Kenntnisse in der Informatik. B den Informatikern verhält es sich meist umgekehrt. Es treffen also Menschen aufeinander, die erst zu einer gemeinsam Sprache und einem gemeinsamen Verständnis finden müssen.
- Es gibt meist nicht *den* Kunden oder Benutzer, sondern sehr viele mit recht unterschiedlichen Anforderungen, die sich nicht selten widersprechen. Bei der Anforderungsanalyse müssen also Konflikte aufgedeckt und Kompromisse geschlossen werden.
- Die Anforderungen werden nicht selten durch politische Entscheidungen beeinflusst. Das aus organisatorischer und technischer Sicht bestmögliche Soll-Konzept kann dann immer noch aus rein politischen Gründen scheitern. Beispielsweise sieht der Abteilungsleiter durch eine Änderung im Arbeitsfluss dank softwaretechnischer Unterstützung plötzlich seinen Einfluss gefährdet und boykottiert die angestrebte Lösung.
- Die Welt ist einem ständigen Wandel unterworfen. Weil sich die Welt ändert, ändern sich notwendigerweise auch die Anforderungen an die Software, die einen Arbeitsfluss dieser Welt automatisiert. Das geschieht sicherlich in der Zeit nach Auslieferung der Software in der so genannten Wartungsphase, d.h. während ihres Einsatzes, aber nicht selten au schon während der Entwicklung. Mit der Anforderungsanalyse ist man also selten fertig. Man sollte deshalb idealerwei wahrscheinliche zukünftige Änderungen vorwegsehen und schon in der initialen Entwicklung berücksichtigen.

Bewusstseinsebenen

- bewusstes Wissen (20-30%)
  - Wissen, über das man sich im Klaren ist oder das in seiner vollen Bedeutung klar erkannt wird
- unbewusstes Wissen ($\leq$40%)
  - Wissen, das sich dem Bewusstsein im Moment nicht darbietet, aber dennoch handlungsbestimmend ist, und potenziell aufgerufen werden kann
- unterbewusstes Wissen
  - unbekannte Wünsche, die erst von außen herangetragen werden müssen, um als Anforderungen erkannt zu werden



Dass Benutzer oft nicht genau wissen, was sie wollen beziehungsweise es nicht sagen, liegt meist nicht an einem Mangel an Kenntnis, Vorstellungsvermögen oder Kooperationsbereitschaft, sondern an der Gestalt menschlichen Wissens. Menschliches Wissen lässt sich unterteilen in drei ungefähr gleich große Bereiche.

Das **bewusste Wissen** ist solches Wissen, über das man sich im Klaren ist oder das in seiner vollen Bedeutung klar erkannt wird. Fragt man einen Radfahrer danach, wie man mit einem Fahrrad eine Kurve fährt, so wird er einem sicherlich antworten, dass man hierfür den Lenker einschlagen muss. Das bewusste Wissen ist durch direkte Fragen unmittelbar zugänglich.

Das **unbewusste** Wissen ist solches Wissen, das sich dem Bewusstsein im Moment nicht darbietet, aber dennoch handlungsbestimmend ist und potenziell aufgerufen werden kann. Wird man denselben Radfahrer darauf aufmerksam machen, dass man beobachtet habe, dass er seinen Körper neigt, bevor er den Lenker zur Kurve einschlägt, wird er die maßgebende Rolle des Körpergewichtes bestätigen. An das unbewusste Wissen vermag man durch Beobachtung und Nachfragen zu kommen.

Während sowohl das bewusste als auch das unbewusste Wissen wenigstens indirekt erschlossen werden kann, kommt man an die dritte Gestalt menschlichen Wissens kaum oder nur sehr schwer heran. Ein Drittel des menschlichen Wissens ist dem Wissenden selbst nämlich gar nicht bewusst. Das **unterbewusste** Wissen besteht aus dem uns nicht bekannten Wissen, das dennoch für unsere Handlungen maßgebend ist. Oft sind es Ängste, unbewusste Wünsche oder Triebe, die für uns handlungsbestimmend sind. Warum der Radfahrer in bestimmten Situationen es scheut, eine Kurve schneller zu nehmen, hat vielleicht damit zu tun, dass er sich einmal kräftig durch einen Sturz blamiert hat, als er einst versuchte, eine Kurve besonders schnittig zu nehmen, um seinen Freunden zu imponieren. Unbewusste Anforderungen müssen erst von außen an uns herangetragen werden, um als Anforderungen erkannt zu werden.

Bewusstseinsebenen

- bewusstes Wissen (20-30%)
  - Wissen, über das man sich im Klaren ist oder das in seiner vollen Bedeutung klar erkannt wird
- unbewusstes Wissen ($\leq$40%)
  - Wissen, das sich dem Bewusstsein im Moment nicht darbietet, aber dennoch handlungsbestimmend ist, und potenziell aufgerufen werden kann
- unterbewusstes Wissen
  - unbekannte Wünsche, die erst von außen herangetragen werden müssen, um als Anforderungen erkannt zu werden

Folgt für den Informatiker, der eine Anforderungsanalyse durchführt, nun daraus, dass er besser eine Ausbildung als Psychoanalytiker absolvieren sollte? Wenngleich das sicherlich helfen könnte, so ist es für unsere Zwecke meist ausreichend, sich über diese Formen des Wissens im Klaren zu sein und passende Techniken anzuwenden, um an möglichst viel Wissen und Anforderungen zu kommen. Mit reinen Fragetechniken wird man zum überwiegenden Teil allein das bewusste Wissen erschließen können. Mit Beobachtungen und Nachfragen kann man noch den Bereich des unbewussten Wissens erfassen. Bei der Einschätzung der Aussagen eines Benutzers sollte man sich aber auch Gedanken machen, warum er eine Antwort in einer bestimmten Weise geben könnte oder warum er sich auf eine bestimme Art und Weise verhält. Es könnten zum Beispiel Ängste vor der Umstellung durch die softwaretechnische Lösung sein, die einen Menschen unbewusst umtreiben – seien es berechtigte Ängste um den möglichen Verlust des Arbeitsplatzes oder einfach nur die allgemeine Angst, durch die Neuerungen überfordert zu werden.

Basisfaktoren

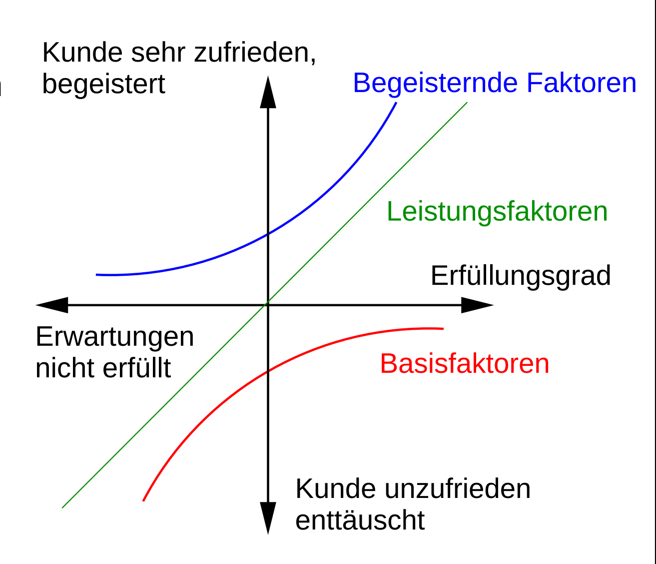
- Minimalanforderungen
- → Mangel führt zu massiver Unzufriedenheit
- → mehr als Zufriedenheit ist nicht möglich

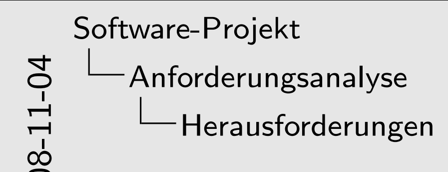
Leistungsfaktoren

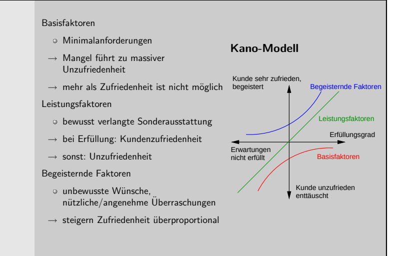
- bewusst verlangte Sonderausstattung
- → bei Erfüllung: Kundenzufriedenheit
- → sonst: Unzufriedenheit

Begeisternde Faktoren

- unbewusste Wünsche, nützliche/angenehme Überraschungen
- → steigern Zufriedenheit überproportional







Eine gute Anforderungsanalyse ist der Grundstock für den Erfolg. Dazu muss es uns gelingen, den Kunden mit unserem Produkt zu begeistern, indem wir auf alle seine Anforderungen eingehen und diese später auch umsetzen.

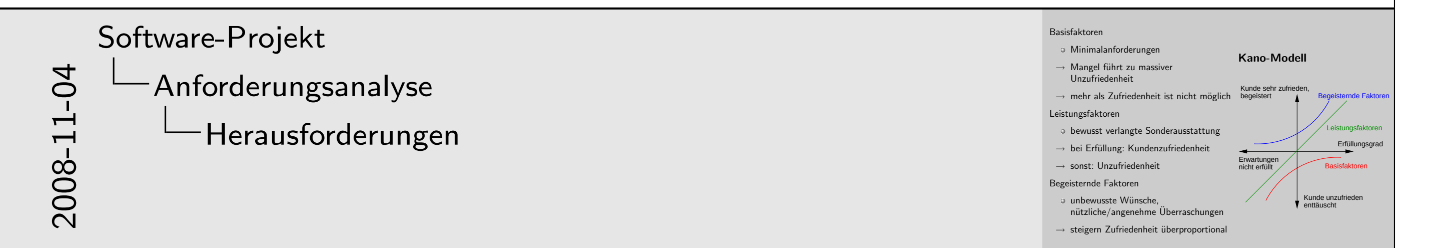
Der Zusammenhang zwischen dem Erfüllungsgrad der Anforderungen und der Begeisterung, die wir damit beim Kunden wecken können, wird durch das Kano-Modell (Kano 1984) beschrieben. Generell gilt offensichtlich, dass wir mit einem höheren Erfüllungsgrad der Anforderungen auch eine höhere Zufriedenheit erreichen können. Ob eine erfüllte Anforderung den Kunden zur Begeisterung treibt oder ob es für ihn vielmehr eine Selbstverständlichkeit darstellt, hängt jedoch ab von der Art der Anforderung.

Die erste Kategorie von Anforderungen sind die **Basisfaktoren** der Software. Diese Anforderungen stellen Mindestanforderungen dar, die die Software erfüllen muss, damit sie eingesetzt werden kann, Werden sie nicht erfüllt, ist die Software unbrauchbar. Dies führt zu massiver Unzufriedenheit. Da es sich um Mindestanforderungen handelt, werden wir den Kunden selbst mit einer hundertprozentigen Erfüllung allenfalls zufrieden, aber nicht begeistern können.

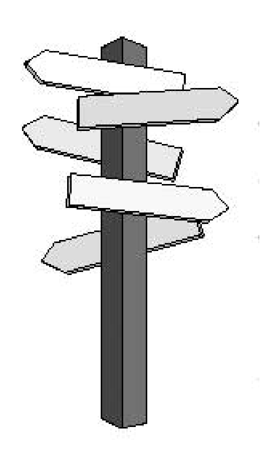
Die Gesamtheit der **Leistungsfaktoren** ist die bewusst verlangte Sonderausstattung. Werden diese Anforderungen erfüllt, können wir den Kunden mehr als nur zufrieden stellen. Werden sie nicht erfüllt, ist der Kunde enttäuscht – er hatte sich diese Leistungsfaktoren ja explizit gewünscht. Es droht dann Unzufriedenheit.

Den Kunden begeistern können wir, indem wir ihn überraschen. Die **begeisternde** Faktoren erfüllen ihm unbewusste Wünsche oder stellen nützliche und angenehme Überraschungen dar. Mit diesen Faktoren können wir die Zufriedenheit überproportional steigern, aber bei Nichterfüllen niemals enttäuschen, weil der Kunde diese Anforderungen nicht explizit verlangt hat.

2008-11-04



Daraus folgt nun nicht, dass wir eigene Anforderungen hinzu erfinden sollten, von denen wir glauben, dass sie den Kunden vom Hocker reißen werden. Schließlich müssen wir die Anforderungen implementieren und wenn sie vom Kunden nicht bestellt sind, müssen wir dafür bezahlen. Wenn es uns aber gelingt, solche Anforderungen in der Anforderungsanalyse zu identifizieren und dem Kunden vorzuschlagen, so können sie zum Vertragsgegenstand werden, für die er gerne Geld ausgibt. Allerdings werden sie so dann schnell zu Leistungsfaktoren, die bei Nichterfüllung Unzufriedenheit hervor rufen. Aber zumindest kann es für den Kunden den Ausschlag geben, sich für unsere Dienste und nicht für die unserer Mitbewerber zu entscheiden.

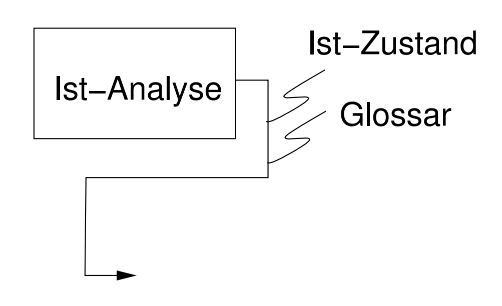


Erhebung der Anforderungen

Was sind die wesentlichen Schritte zur Erhebung der Anforderungen?

Wegweiser
Erhebung der Anforderungen
Was sind die wesentlichen Schritte zur Erhebung der Anforderungen?

In diesem Abschnitt wenden wir uns den einzelnen Aktivitäten der Anforderungsanalyse zu. Dazu gehören die Ist-Analyse, die Soll-Analyse und das Festhalten der Anforderungen in Form einer Anforderungsspezifikation. Diese Aktivitäten haben eine natürlichen Abfolge, weil sie voneinander kausal abhängen. Allerdings werden sie in der Praxis meist wiederholt.



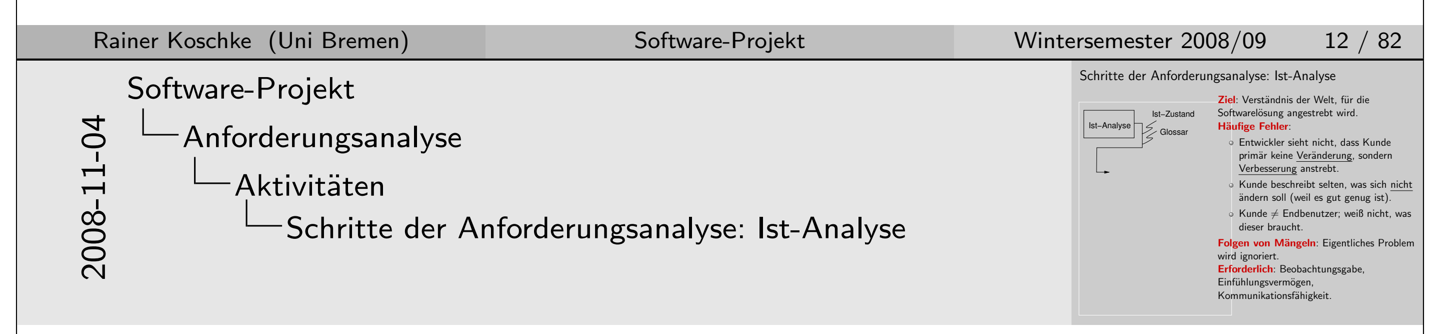
Ziel: Verständnis der Welt, für die Softwarelösung angestrebt wird.

Häufige Fehler:

- Entwickler sieht nicht, dass Kunde primär keine Veränderung, sondern Verbesserung anstrebt.
- Kunde beschreibt selten, was sich nicht ändern soll (weil es gut genug ist).
- Kunde $\neq$ Endbenutzer; weiß nicht, was dieser braucht.

Folgen von Mängeln: Eigentliches Problem wird ignoriert.

Erforderlich: Beobachtungsgabe, Einfühlungsvermögen, Kommunikationsfähigkeit.

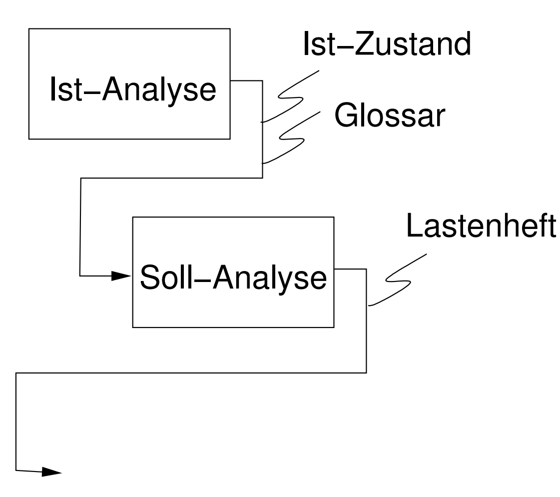


Die Aktivität **Ist-Analyse** zielt darauf ab, den Ausschnitt der Welt zu verstehen, für den eine softwaretechnische Lösung angestrebt wird. Das heißt, wir streben an, den Ist-Zustand genau zu verstehen und zu analysieren. Über die Beschreibung des Ist-Zustands hinaus, entsteht in dieser Phase ein Glossar (Begriffslexikon), in dem wir die Begriffe aus der Anwendungsdomäne definieren, die der Kunde verwendet. Das Begriffslexikon ist notwendig, um Unklarheiten und Missverständnisse auszuschließen. In der Regel sind Softwareentwickler mit den Begriffen der Anwendungsdomäne nicht vertraut genug oder haben eine etwas falsche Vorstellung. Besonders problematisch sind die Fälle, in denen wir meinen zu wissen, was vermeintlich gemeint ist, aber leider daneben liegen, ohne es zu merken. Dieses Glossar wird vom Kunden gelesen und verifiziert. Es wird über die Projektdauer gepflegt und angepasst.

Ein häufiger Fehler bei der Ist-Analyse ist es, dass wir uns nicht bewusst werden, dass der Kunde primär keine Veränderung, sondern eine Verbesserung anstrebt. Das bedeutet, dass der Kunde mit vielen Dingen so wie sie sind bereits zufrieden ist und die auch weiterhin so bleiben sollen. Der Kunde beschreibt aber selten, was sich nicht ändern soll, weil es gut genug ist. Er konzentriert sich vielmehr darauf, was sich ändern soll. Nichtsdestotrotz müssen wir auch die Anforderungen kennen, die übernommen werden sollen. Wenn es darum geht, ein existierendes Softwaresystem abzulösen, bietet sich uns eine große Chance vom existierenden System zu lernen. Viele Informatiker machen jedoch den Fehler, das existierende Systeme allerhöchstens oberflächlich zu untersuchen. Dabei könnten wir den Kunden gezielt danach fragen, was wir vom existierenden System übernehmen beziehungsweise verbessern sollten.

Erfolgt die Ist-Analyse mangelhaft, droht die Gefahr, dass wir das eigentliche Problem ignorieren. Für eine erfolgreiche Durchführung der Ist-Analyse benötigen wir Beobachtungsgabe, Einfühlungsvermögen und Kommunikationsfähigkeit.

Schritte der Anforderungsanalyse: Soll-Analyse



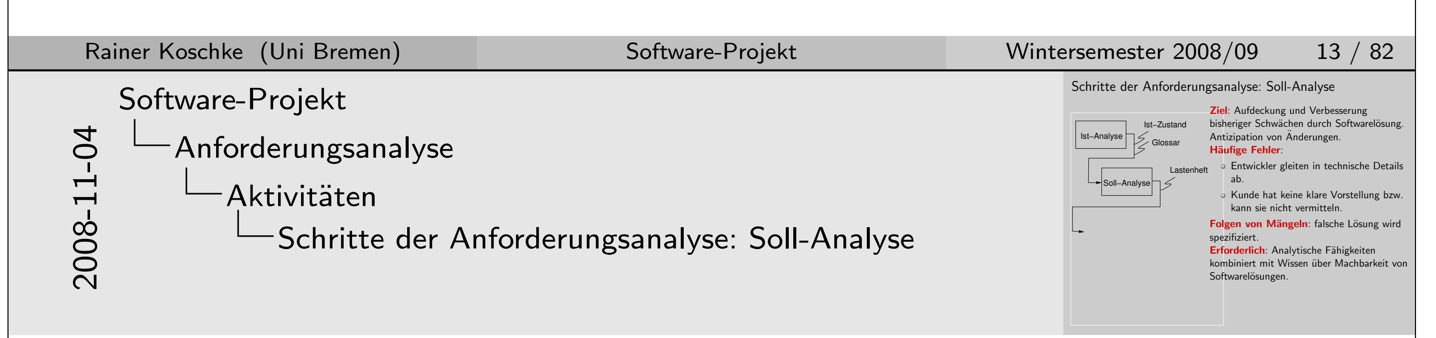
Ziel: Aufdeckung und Verbesserung bisheriger Schwächen durch Softwarelösung. Antizipation von Änderungen.

Häufige Fehler:

- Entwickler gleiten in technische Details ab.
- Kunde hat keine klare Vorstellung bzw. kann sie nicht vermitteln.

Folgen von Mängeln: falsche Lösung wird spezifiziert.

Erforderlich: Analytische Fähigkeiten kombiniert mit Wissen über Machbarkeit von Softwarelösungen.



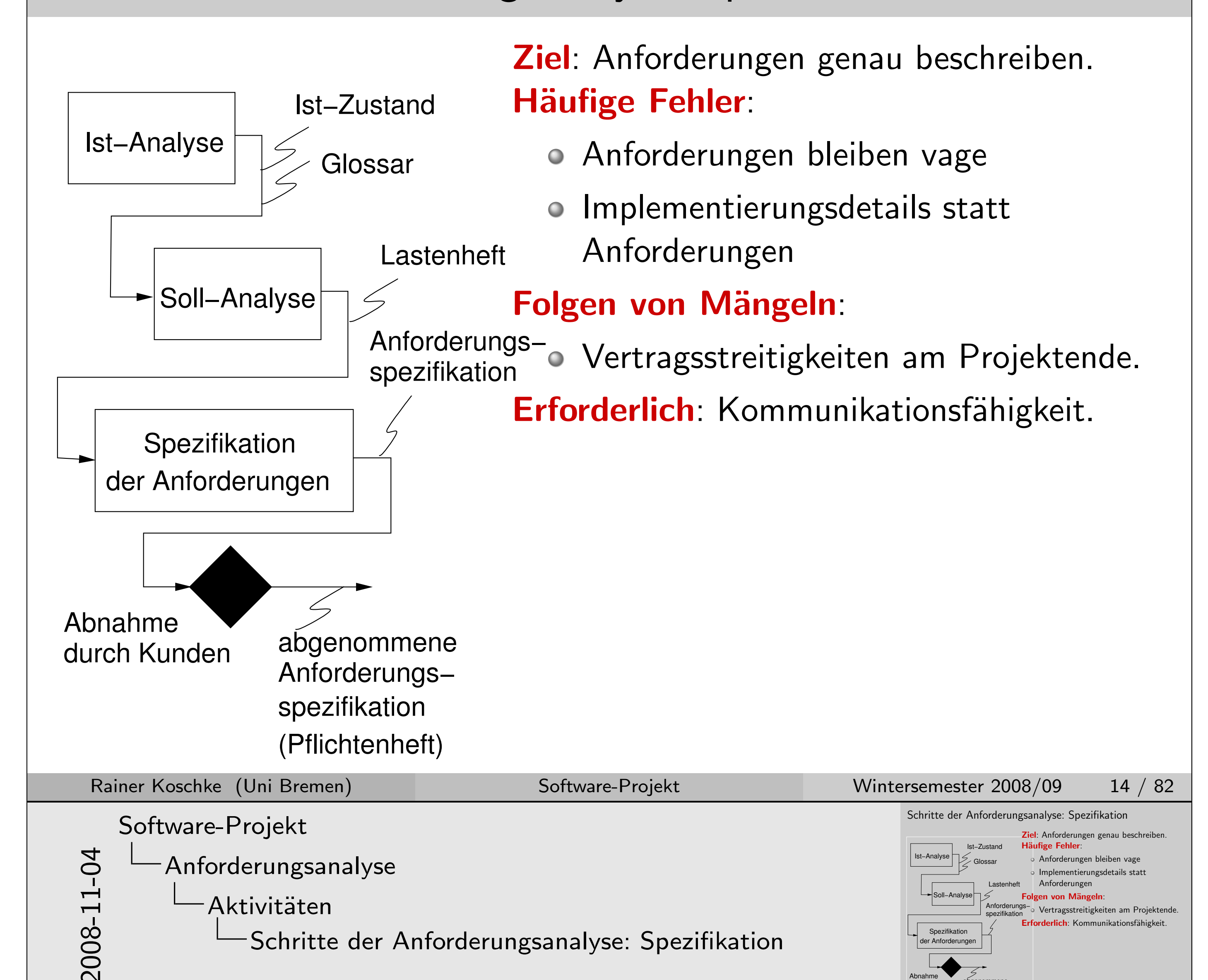
Auf Basis der Ergebnisse der Ist-Analyse identifiziert die **Soll-Analyse** die bisherigen Schwächen und erarbeitet Verbesserungsmaßnahmen. Dabei interessieren uns nicht nur die unmittelbar anstehenden Anforderungen sondern auch jene, die in absehbarer Zukunft relevant werden können. Wahrscheinliche zusätzliche Anforderungen an das System, die in der Zukunft eintreffen werden, sollen beschrieben werden, damit die Softwarelösung so strukturiert werden kann, dass diese Änderungen später einfach zu realisieren sind.

Das Ergebnis der Soll-Analyse ist das Lastenheft, in dem wir aus Sicht des Kunden beschreiben, was die Probleme der bisherigen Situation sind und was die neue Software verbessern soll.

Häufig wird in der Soll-Analyse der Fehler gemacht, dass wir in technische Details der Lösung abgleiten, obwohl es doch zunächst um grobe Konzepte zur Verbesserung geht. Wir stoßen auch auf das oben bereits angeführte Problem, dass der Kunde nicht notwendigerweise eine klare Vorstellung von der möglichen softwaretechnischen Verbesserung hat beziehungsweise sie nicht vermitteln kann.

Mängel in der Soll-Analyse führen dazu, dass eine falsche Lösung spezifiziert wird. Am Ende entsteht ein Softwaresystem, das nicht die relevanten Probleme beseitigt und damit unnütz ist. Um eine gute Soll-Analyse machen zu können, brauchen wir neben analytischen Fähigkeiten bei der Problemsuche auch softwaretechnisches Wissen, um abzuschätzen ob und mit welchem Aufwand, anvisierte softwaretechnische Lösungen machbar sind.

Schritte der Anforderungsanalyse: Spezifikation



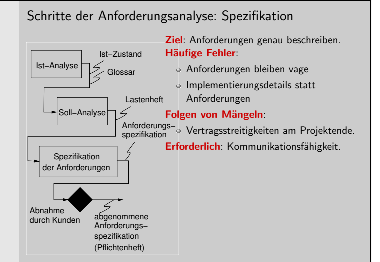
Die Ist-Analyse untersucht die Anwendungsdomäne, für die eine softwaretechnische Lösung angestrebt wird. Diese Analyse ist rein deskriptiv. Die Soll-Analyse untersucht die Schwächen des Ist-Zustands und formuliert daraus Anforderungen an eine softwaretechnische Lösung, die diese Schwächen ausgleichen sollen. Die Soll-Analyse muss Prioritäten setzen (nicht alle Anforderungen sind gleich wichtig) und mögliche Konflikte auflösen (Anforderungen können sich widersprechen; z.B. Performanz versus Sicherheit). Bei der Soll-Analyse müssen eventuell auch Folgen von softwaretechnischen Lösungen abgeschätzt werden. Diese Abschätzung könnte dazu führen, eine andere softwaretechnische Lösung zu finden beziehungsweise – im Extremfall – eine softwaretechnische Lösung ganz auszuschließen.

Mit *Lösung* ist hier nicht die Implementierung gemeint, sondern lediglich der Anforderungskatalog an die Implementierung. Es geht hier noch nicht darum, sich zu überlegen, wie die Anforderungen implementiert werden.

Das Lastenheft beschreibt die Wünsche des Kunden aus dessen Sicht. Es ist häufig noch unscharf oder unstimmig. Erst die Anforderungsspezifikation (auch Pflichtenheft) beschreibt genau, was zu implementieren ist.

Obwohl die Anforderungsspezifikation nur beschreiben soll, *was* implementiert und nicht *wie* implementiert werden soll, sind Überlegungen zu der generellen Machbarkeit notwendig. Eine Anforderungsspezifikation muss auch realisierbar sein. Dennoch sollten diese Überlegungen lediglich dazu führen, möglicherweise die Anforderungen abzuändern. Mögliche Wege, die Anforderungen zu implementieren, gehören in ein Entwurfsdokument und nicht in die Anforderungsspezifikation.

Die Anforderungsspezifikation stellt den Vertrag zwischen Kunde und Entwickler dar und muss entsprechend genau geprüft werden. Meist geschieht dies durch ein Review mit dem Kunden.

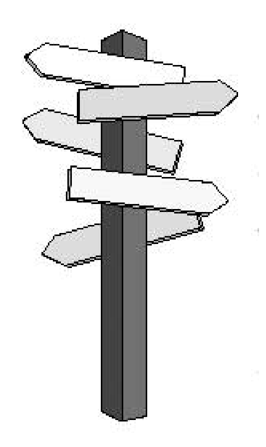


Typische Fehler bei der Anforderungsspezifikation sind vage Anforderungen (z.B. dass die Software *benutzerfreundlich* sein soll) und dass sie Implementierungsdetails beschreibt statt Anforderungen aus Kundensicht.

Weil die Anforderungsspezikation den Vertragsgegenstand festlegt, führen Mängel nicht selten zu Vertragsstreitigkeiten am Ende des Projekts. Wenn die Anforderungen nicht genau definiert wurden, ist eine gerichtliche Auseinandersetzung sehr mühselig. In jedem Falle ist ein solcher Streit für alle Parteien zum eigenen Schaden. Der Kunde hat nicht bekommen, was er wollte; wir bekommen möglicherweise unser Geld nicht. Ein enttäuschter Kunde schadet uns, denn Enttäuschung spricht sich schnell herum.

Wir werden uns also große Mühen bei der Anforderungsspezifikation geben. In den folgenden Abschnitten werden wir hierzu verschiedene Techniken kennen lernen, die uns bei richtiger Ausführung vor Schaden bewahren können. Es sei auch hier nochmal darauf hingewiesen, dass die genannten Schritte der Ist- und Soll-Analyse sowie das Festhalten der Anforderungen hochgradig iterativ sind und keineswegs so sequentiell, wie durch den Datenfluss im Diagramm suggeriert. Das muss bei der Planung beachtet werden.

Wegweiser



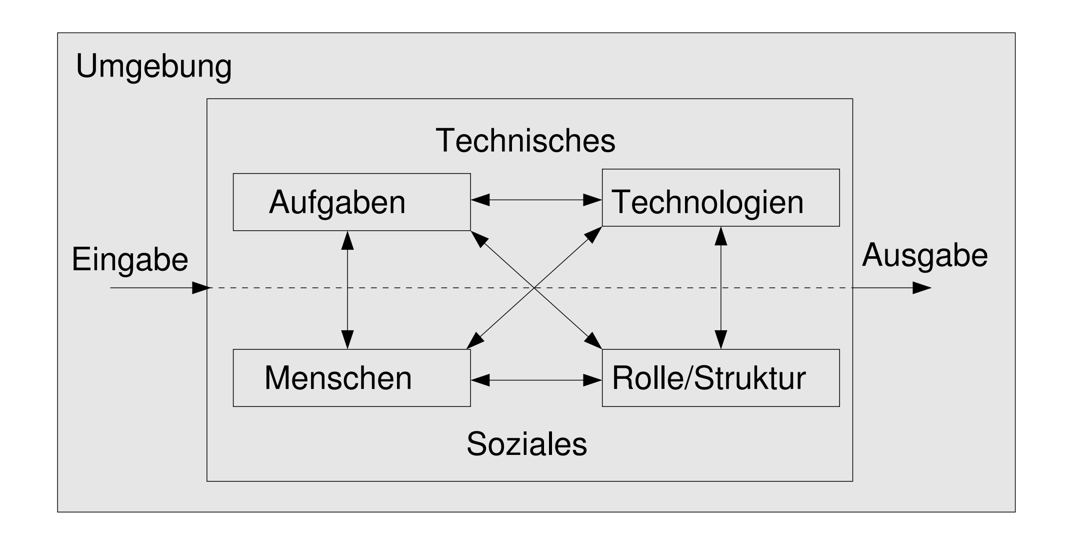
Ist-Analyse

Was ist das Ziel und der Gegenstand der Ist-Analyse?

Soziotechnisches System

Definition

Soziotechnisches System: organisierte Menge von Menschen und Technologien, die in einer bestimmten Weise strukturiert sind, um eine Aufgabe zu erfüllen.



– Emery, Thorsrud & Trist 1964

Ist-Zustand: Was?

Verständnis von:

- Struktur
- Aufgaben
- Kommunikation
- Dokumenten
- Daten
- Schwachstellen

. . . aus allen relevanten Blickwinkeln.

Ist-Zustand: Was?
Verständnis von:
- Struktur
- Aufgaben
- Kommunikation
- Dokumenten
- Daten
- Schwachstellen

... aus allen relevanten Blickwinkeln.

In diesem Abschnitt lernen wir Analysetechniken für die Ist-Analyse kennen. Zunächst untersuchen wir, was wir bei der Ist-Analyse überhaupt analysieren sollen.

Durch die Ist-Analyse streben wir ein Verständnis folgender Aspekte des Ist-Zustands aus allen relevanten Blickwinkeln an:

- **Struktur**: das organisatorische Gefüge des Systems, für das die Softwarelösung angestrebt wird.
- **Aufgaben**: der Umfang und die Art der anfallenden Aufgaben (Operationen) und Besonderheiten im Ablauf
- **Kommunikation**: die Vorrichtungen und Gelegenheiten zur Kommunikation und ihr Ablauf
- **Dokumenten:** schriftliche oder elektronische Dokumente, die verwendet und produziert werden
- **Daten**: Umfang und Art der verarbeiteten Daten in den Dokumenten und über sie hinaus
- **Schwachstellen**: die bestehenden Mängel, Unvollständigkeiten und Redundanzen des Ist-Zustands

Wie werden diese Aspekte nun vertiefen und benutzen als illustrierendes Beispiel eine Verkaufssituation in einem Fahrradladen, für die wir eine softwaretechnische Unterstützung entwickeln sollen.

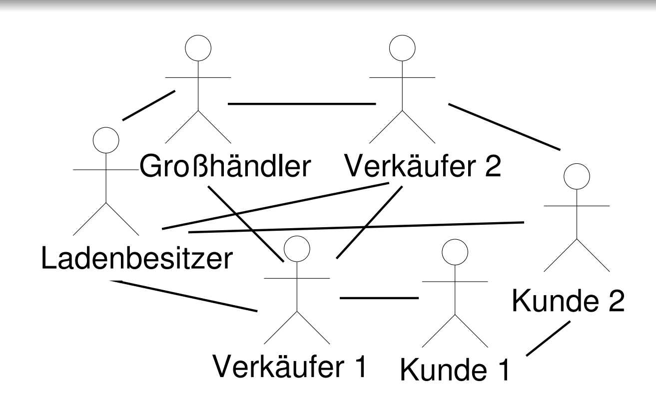
Ist-Zustand: Was?

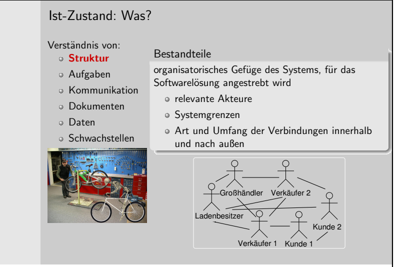
Verständnis von:
- **Struktur**
- Aufgaben
- Kommunikation
- Dokumenten
- Daten
- Schwachstellen

Bestandteile

organisatorisches Gefüge des Systems, für das Softwarelösung angestrebt wird
- relevante Akteure
- Systemgrenzen
- Art und Umfang der Verbindungen innerhalb und nach außen







Bei der Analyse der Struktur identifizieren wir zunächst alle relevanten Akteure (Benutzer und andere Softwaresysteme, die miteinander interagieren. Im Fahrradladen sind das Kunden und Verkäufer sowie der Ladenbesitzer und seine Lieferanten.

Dann legen wir die Systemgrenzen fest. Mit System ist hier weder das Softwaresystem gemeint, das wir implementieren sollen (wir kennen ja die Anforderungen dafür noch gar nicht) noch etwa ein möglicherweise bereits existierendes Softwaresystem, das wir ablösen sollen. Vielmehr meinen wir damit den Ausschnitt der Welt, für den wir die softwaretechnische Verbesserung anstreben sollen. Diesen Ausschnitt fassen wir als ein System miteinander agierender Akteure auf.

Wenn wir die Systemgrenzen festlegen, bestimmen wir also, was alles zu dem Ausschnitt der Welt gehört, der für uns relevant ist. Bsp.: *Nehmen wir an, wir sind beauftragt, das Verkaufsgespräch zwischen Käufer und Verkäufer zu optimieren, dann gehören diese beiden Personengruppen zu unserem untersuchten System. Ihr Zusammenspiel sollen wir optimieren. Ladenbesitzer und Lieferanten hingegen sind außerhalb unseres Systems, aber dennoch relevant für unsere Aufgabenstellung, weil der Verkäufer den Ladenbesitzer bei bestimmten Entscheidungen zu Rate ziehen und Lieferanten zu Lieferzeiten befragen muss.* □

Haben wir die Systemgrenzen gezogen, untersuchen wir die Art und den Umfang der Verbindungen innerhalb des Systems und nach außen. Wir bestimmen, wer innerhalb des System miteinander und welche unserer inneren Akteure mit welchen externen Akteuren interagiert. Die Interaktion zwischen externen Akteuren können wir in der Regel vernachlässigen.

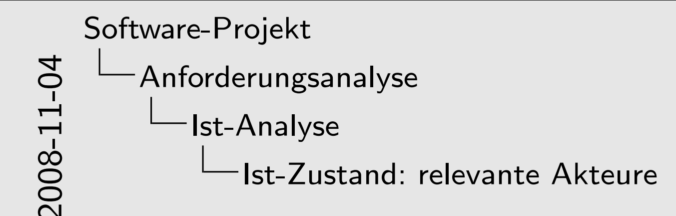
Das Ergebnis lässt sich als Graph repräsentieren, dessen Knoten die Akteure und dessen Kanten die Interaktionen darstellen.

Ist-Zustand: relevante Akteure

Grundsatz: Kenne deinen Benutzer!

Aber: *Der* Benutzer bzw. *die* Benutzerin ist eine Illusion. Es sind individuelle Menschen, um die es geht.

Andererseits: Wir können nicht jeden betrachten und müssen zusammenfassen.



Grundsatz: Kenne deinen Benutzer!

Aber: *Der* Benutzer bzw. *die* Benutzerin ist eine Illusion. Es sind individuelle Menschen, um die es geht.

Andererseits: Wir können nicht jeden betrachten und müssen zusammenfassen.

Nachdem wir alle Akteure bestimmt haben, untersuchen wir die menschlichen Akteure weiter im Detail. Dies sind die potentiellen Benutzer unseres zukünftigen Softwaresystems. Zum Teil erscheinen sie uns bisher noch recht abstrakt, etwa *der Verkäufer* und *der Käufer*. Diese Rollen definieren bestimmte Klassen von Benutzern mit unterschiedlichen Aufgaben und Zielen in unserem Systemgefüge. Diese Rollen abstrahieren aber von den Eigenschaften konkreter Benutzer. So können wir sicherlich ganz unterschiedliche Käufer in unserem Fahrradladen ausmachen, die sich in Erfahrungen, Vorlieben, Wünschen, Alter, Geschlecht und Finanzkraft unterscheiden. Alle wollen ein Fahrrad oder ein Fahrradbestandteil kaufen, aber ihre Auswahl und ihre Interaktion mit dem Käufer wird wesentlich durch ihre persönlichen Eigenschaften bestimmt. Es reicht also nicht aus, nur funktionale Rollen zu identifizieren. Da das Softwaresystem von konkreten Menschen benutzt werden wird, müssen wir deren Eigenheiten genau kennen, damit wir ein nützliches Softwaresystem für sie entwickeln können.

Andererseits können wir auch nicht jedes mögliche Individuum einzeln betrachten, da die Anzahl der Benutzer meist sehr hoch ist und uns nicht alle Benutzer bekannt sind. Aus ökonomischen Gründen müssen wir also mehrere Individuen mit ähnlichen Eigenschaften zu Klassen zusammenfassen.

Ist-Zustand: relevante Akteure

Persona

(in archetypischer Psychologie) die Maske oder Erscheinung, die man der Welt präsentiert.

(in der Softwareergonomie) erzählerische Beschreibung charakterischer Eigenschaften und Verhalten eines Benutzers oder Kunden, die spezifische Details nennt, statt Verallgemeinerungen.



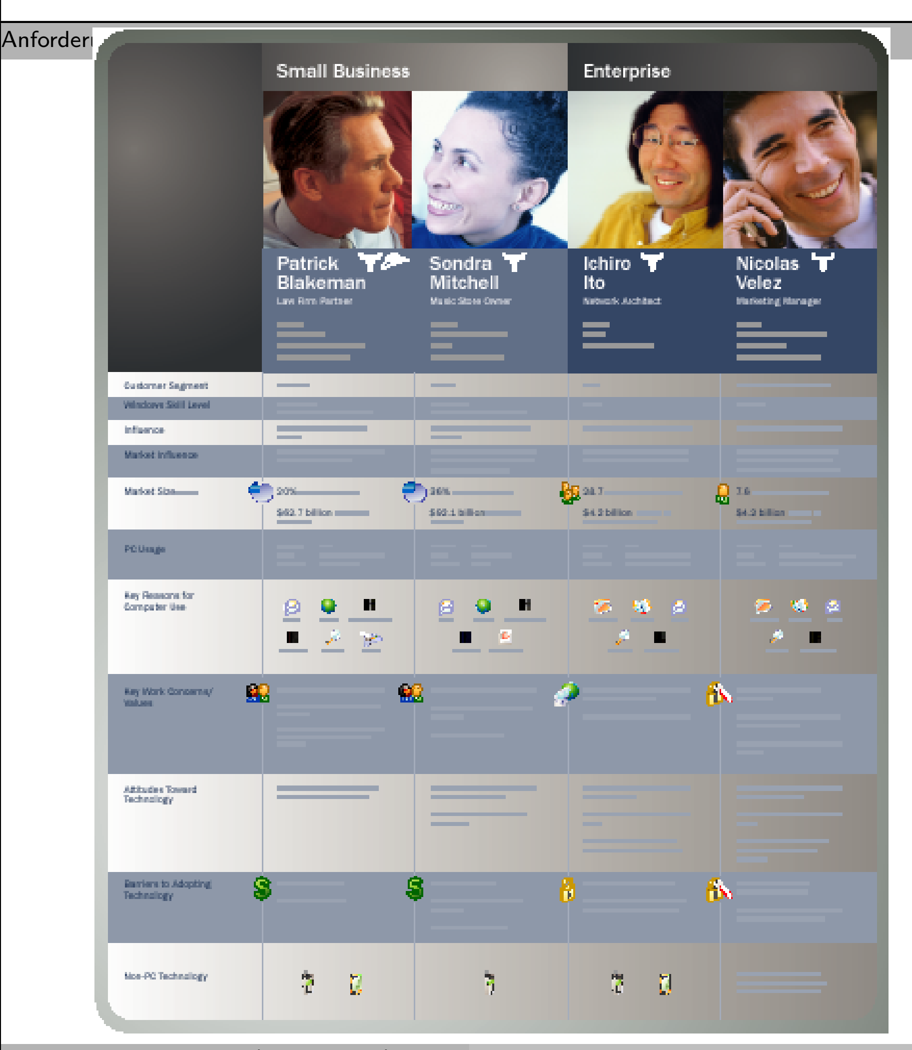
Ist-Zustand: relevante Akteure

Persona
(in archetypischer Psychologie) die Maske oder Erscheinung, die man der Welt präsentiert.
(in der Softwareergonomie) erzählerische Beschreibung charakteristischer Eigenschaften und Verhalten eines Benutzers oder Kunden, die spezifische Details nennt, statt Verallgemeinerungen.

Es ist die Aufgabe des Analysten, eine angemessene Menge von Benutzerklassen zu finden, die die richtige Balance zwischen abstrakten Rollen und individuellen Benutzern wahrt. Zur Beschreibung dieser Benutzerklassen kann man sich der so genannten Personas bedienen. Die **Persona** ist ursprünglich eine im griechischen Theater von den Schauspielern verwendete Maske, die die Rolle typisierte und als Schallverstärker benutzt wurde. In der archetypischen Psychologie ist es die Maske oder Erscheinung, die man der Welt präsentiert.

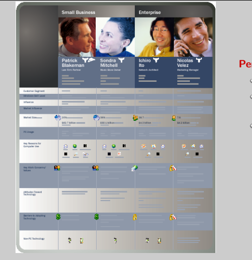
In der Anforderungsanalyse der Softwaretechnik steht eine Persona für eine Benutzerklasse. Sie beschreibt deren gemeinsamen Eigenschaften, gibt ihnen aber ein konkretes Bild. Sie ist in der Softwareergonomie eine erzählerische Beschreibung charakterischer Eigenschaften und Verhalten eines Benutzers oder Kunden, die spezifische Details nennt, statt nur Verallgemeinerungen.

Neben der Konkretisierung statt Verallgemeinerung geben uns die Personas etwas Spielerisches im Entwicklungsprozess, das unsere Kreativität stimulieren kann. Wir können große Poster unserer Personas aufhängen und sie betrachten, während wir an der Anforderungsspezfikation schreiben. Wir leben damit die Idee, für und mit dem Benutzer zu entwickeln.



Persona-Poster:

- Bild (fiktiv)
- Name (fiktiv), Beruf, Motto
- Rolle, Ziele, Aufgaben, Ideen, Wünsche, Vorlieben, persönliche Details



Persona-Poster:

- Bild (fiktiv)
- Name (fiktiv), Beruf, Motto
- Rolle, Ziele, Aufgaben, Ideen, Wünsche, Vorlieben, persönliche Details

Die ursprüngliche Idee der Personas für die Anforderungsaanalyse stammt von Cooper. Zur Beschreibung einer Persona gehören ein fiktives Bild und ein fiktiver Name sowie weitere persönliche Eigenschaften, wie den Beruf oder ein charakterisierendes Motto. Statt von der abstrakten Benutzerklasse *Verkäufer* sprechen wir also von *Herrn Schmidt*, dem Verkäufer. Dann werden die gemeinsamen Eigenschaften der Benutzer, die sich hinter der Persona verbergen, aufgeführt. Dazu gehören die Rolle, die Herr Schmidt ausfüllt, die Ziele, die Herr Schmidt verfolgt, seine Ideen, Wünsche, Vorlieben und weitere persönliche Details, soweit sie von Relevanz für die Analyse sind.

Quelle für das Bild:
`http://www.research.microsoft.com/research/coet/Grudin/Personas/Pruitt-Grudin.doc`. Der Artikel erläutert auch wesentliche Ideen.

Bedeutung von Personas

- archetypische Benutzerbeschreibungen
- typisch für Zielgruppen
- decken deren Anforderungen, Bedürfnisse und Ziele ab
- stehen im Designprozess stellvertretend für die realen Benutzer (statt der relativ anonymen und pauschalen Größe "Benutzer")
- können bei Design und beim Usability-Test der Benutzerinteraktion verwendet werden
- können beim Handbuchschreiben und -prüfen sowie Akzeptanztest verwendet werden

– Astrid Beck, FHT Esslingen

Bedeutung von Personas

- archetypische Benutzerbeschreibungen
- typisch für Zielgruppen
- decken deren Anforderungen, Bedürfnisse und Ziele ab
- stehen im Designprozess stellvertretend für die realen Benutzer (statt der relativ anonymen und pauschalen Größe "Benutzer")
- können bei Design und beim Usability-Test der Benutzerinteraktion verwendet werden
- können beim Handbuchschreiben und -prüfen sowie Akzeptanztest verwendet werden

– Astrid Beck, FHT Esslingen

Astrid Beck, von der Fachhochschule für Technik in Esslingen, fasst die Bedeutung von Personas in der Softwaretechnik wie folgt zusammen. Personas sind archetypische Benutzerbeschreibungen und beschreiben typische Zielgruppen durch deren gemeinsame Eigenschaften. Die Gesamtheit der Anforderungen aller Personas deckt die Anforderungen, Bedürfnisse und Ziele aller Benutzer ab. Wie viele Personas man benötigt, hängt stark vom Projekt ab. In der Regel wird man mit zwischen 5 und 15 verschiedene Personas auskommen.

Die Personas stehen im Designprozess stellvertretend für die realen Benutzer und bilden eine kreative und konkrete Alternative zu der relativ anonymen und pauschalen Größe „Benutzer". Die Anforderungsspezifikation muss die Anforderungen für jede Persona beinhalten. Sie können aber nicht nur während der Anforderungsanalyse der funktionalen Anforderungen an die Software verwendet werden. Die Personas helfen uns auch beim Design der Mensch-Maschine-Kommunikation und beim Usability-Test der Benutzerinteraktion. Hier muss die Benutzerinteraktion so gestaltet sein, dass alle Personas ihr Anwendungsproblem mit dem Softwaresystem effektiv und effizient lösen können. Aus den Personas lassen sich für den Usability-Test, der dies überprüfen soll, unmittelbar entsprechende Test-Benutzer ableiten, die das Profil der Personas erfüllen. Schließlich können die Personas beim Handbuchschreiben und -prüfen sowie beim Akzeptanztest verwendet werden. Auch hier helfen sie, sicher zu stellen, dass die Anforderungen zutreffend und vollständig sind, indem das Handbuch und System aus allen relevanten Blickwinkeln der Benutzersicht untersucht wird.

Ist-Zustand: Was?

Verständnis von:

- Struktur
- **Aufgaben**
- Kommunikation
- Dokumenten
- Daten
- Schwachstellen

Bestandteile

Umfang und Art der anfallenden Aufgaben (Operationen) und Besonderheiten im Ablauf.

- Was wird gemacht?
  → *Kundenberatung*
- Wer oder was führt Operation aus?
  → *Kunde + Verkäufer*
- Wann und wie häufig?
  → *werktags: 50 Mal/Tag; samstags: 80 Mal/Tag*

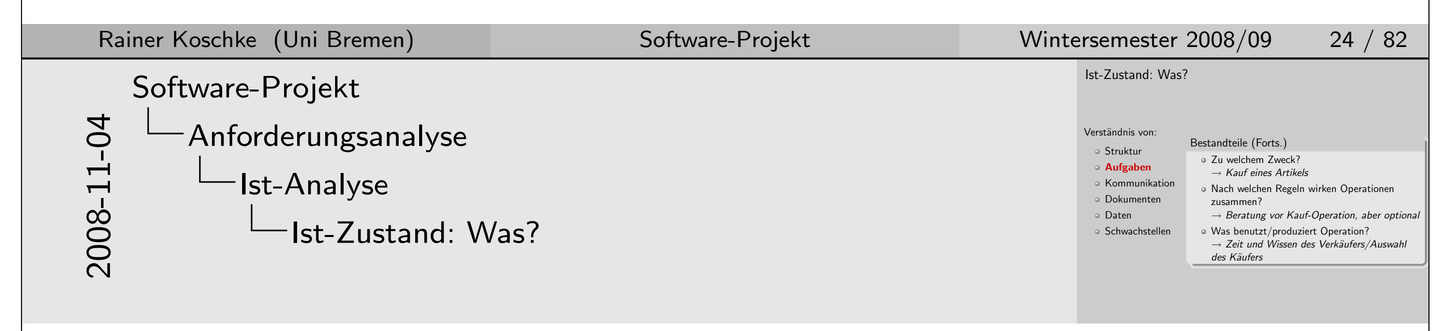
Ist-Zustand: Was?

Verständnis von:

- Struktur
- **Aufgaben**
- Kommunikation
- Dokumenten
- Daten
- Schwachstellen

Bestandteile (Forts.)

- Zu welchem Zweck?
  → *Kauf eines Artikels*
- Nach welchen Regeln wirken Operationen zusammen?
  → *Beratung vor Kauf-Operation, aber optional*
- Was benutzt/produziert Operation?
  → *Zeit und Wissen des Verkäufers/Auswahl des Käufers*



Nachdem wir uns durch die Personas einen Überblick verschafft haben, welche Benutzerklassen zu beachten sind, untersuchen wir als nächstes die Aufgaben der einzelnen Akteure, die im untersuchten System vorkommen. Die Aufgaben können wir als Operationen der Akteure auffassen. Hier interessiert uns, was eine Operation macht, wer sie ausführt, wann und wie häufig die Operation ausgeführt wird, zu welchem Zweck sie durchgeführt wird, welche Regeln der Ausführung der Operation selbst und welche dem Zusammenwirken der Operationen zu Grunde liegen sowie was eine Operation benutzt und produziert.

Ist-Zustand: Was?

Verständnis von:
- Struktur
- **Aufgaben**
- Kommunikation
- Dokumenten
- Daten
- Schwachstellen

Bestandteile (Forts.)
- Zu welchem Zweck?
  → *Kauf eines Artikels*
- Nach welchen Regeln wirken Operationen zusammen?
  → *Beratung vor Kauf-Operation, aber optional*
- Was benutzt/produziert Operation?
  → *Zeit und Wissen des Verkäufers/Auswahl des Käufers*

Betrachten wir diese Fragen zu den Aufgaben an unserem Beispiel des Fahrradladens. Wir untersuchen hier die besondere Aufgabe des Einkaufes. Dies ist eine Operation unter vielen in einem Fahrradladen. Die Reparatur oder der Umtausch sind weitere Beispiele für Operationen in einem Fahrradladen.

Bsp.: *Der Einkauf folgt im Wesentlichen dem folgenden Ablauf:*

1. *Der Kunde betritt den Laden und äußert seinen Wunsch gegenüber dem Verkäufer.*
2. *Der Verkäufer fragt nach den Details und berät den Kunden.*
3. *Der Kunde wählt aus dem Angebot aus, kauft und bezahlt, wenn er etwas Passendes gefunden hat.*
4. *Der Verkäufer übergibt den Artikel und den Kassenzettel, sobald der Betrag durch den Kunden beglichen wurde.*

- *Wer oder was führt Operation aus?*

→ *Die Ausführenden sind Kunde und Verkäufer in den oben beschriebenen Rollen.*
- *Wann und wie häufig?*

→ *Werktags findet diese Operation ca. 50 Mal am Tag statt; samstags ca. 80 Mal/Tag. Abends findet sie tendenziell häufiger statt als morgens.*
- *Zu welchem Zweck?*

→ *Der Käufer benötigt einen Fahrradartikel. Der Verkäufer möchte Umsatz erzielen.*
- *Nach welchen Regeln wirken Operationen zusammen?*

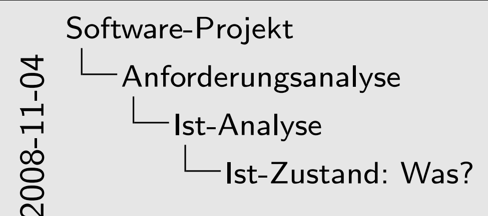
→ *Wenn der Kunde nichts Passendes findet oder der Verkäufer ihn unzureichend berät, geht der Kunde ohne zu kaufen. Wenn der Kunde nicht komplett bezahlen kann, schreibt der Verkäufer an, wenn er den Kunden gut genug kennt. Ansonsten übergibt ihm der Verkäufer den Artikel nicht. Ein Umtausch ist nur möglich, wenn ein Artikel vorher gekau wurde. Käufer und Verkäufer verhalten sich kooperativ, das heißt, der Verkäufer berät nach bestem Wissen und Gewissen und der Kunde liefert ihm hierfür notwendige Hintergrundinformationen zu seinen bereits gekauften Artikeln und seinen Wünschen.*
- *Was benutzt/produziert Operation?*

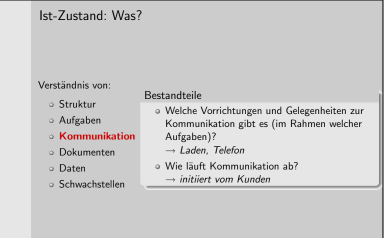
→ *Die Operation überführt den Artikel aus dem Besitz des Ladens in den Besitz des Käufers. Sie überführt den Geldbetra der zu entrichten ist, vom Kunden in die Kasse des Verkäufers.*

Ist-Zustand: Was?

Verständnis von:
- Struktur
- Aufgaben
- **Kommunikation**
- Dokumenten
- Daten
- Schwachstellen

Bestandteile
- Welche Vorrichtungen und Gelegenheiten zur Kommunikation gibt es (im Rahmen welcher Aufgaben)?
  → *Laden, Telefon*
- Wie läuft Kommunikation ab?
  → *initiiert vom Kunden*





Wenn wir die Kommunikation untersuchen, interessieren wir uns für die Vorrichtungen für die Kommunikation, die Gelegenheit, zu der die Kommunikation stattfindet, sowie den Ablauf der Kommunikation.

Bsp.: *Beim Verkaufsgespräch, wie oben dargestellt, verwenden die Akteure im Laden schlicht die menschliche Stimme, wenn das Gespräch im Laden stattfindet. Alternativ kann der Kunde sich aber auch über das Telefon vom Verkäufer beraten lassen und erst später den Laden betreten, um den passenden Artikel zu besorgen.*

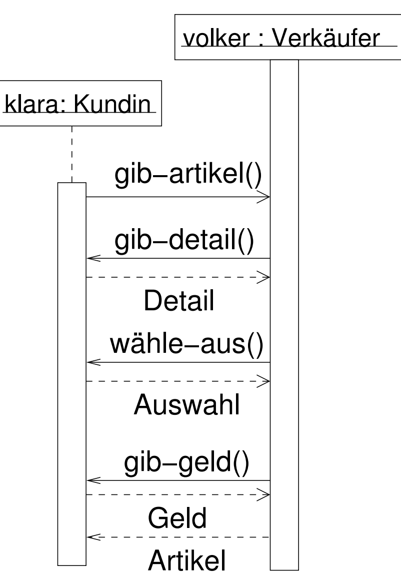
Die Kommunikation findet während der Ladenöffnungszeiten statt und wird vom Kunden initiiert. Wenn der Verkäufer frei ist, spricht er jedoch auch Kunden an, wenn er den Eindruck hat, dass sie seine Hilfe benötigen. □

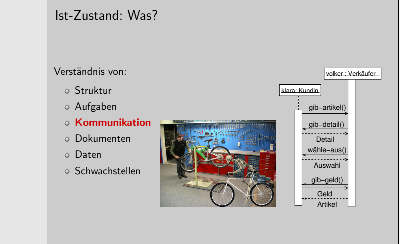
Ist-Zustand: Was?

Verständnis von:

- Struktur
- Aufgaben
- **Kommunikation**
- Dokumenten
- Daten
- Schwachstellen







Den Ablauf der Kommunikation kann man verbal beschreiben oder zu etwas formaleren Notationen greifen. Hierfür eignen sich insbesondere die UML-Sequenzdiagramme, die für solche Zwecke entworfen wurden.

Wichtig ist bei allen gewählten Notationen, dass sie einerseits präzise genug sind – was für stark formalisierte Notationen spricht – und dass sie auch vom Kunden verstanden werden – was meist gegen stark formalisierte Notationen spricht. Sequenzdiagramme sind jedoch ein recht einfach zu erläuterndes Mittel, das nach kurzen Erläuterungen auch Nichtinformatikern verständlich sein kann. Für zustandsbasierte Kommunikation eignen sich auch Zustandsdiagramme. Ebenso kann man mit Aktivitätsdiagrammen komplexere Abläufe gut beschreiben.

Bsp.: *Das Diagramm hier beschreibt die Kommunikation als eine Abfolge von Nachrichten, die sich die Akteure gegenseitig zuschicken. Einige davon, wie z.B. gib-detail(), erhalten eine Antwort. Es bescheibt dieselbe Abfolge wie der natürlichsprachliche Text weiter oben. Das Sequenzdiagramm muss aber hierfür noch ergänzt werden um die Bedeutung der Nachrichten.* □

Ist-Zustand: Was?

Verständnis von:

- Struktur
- Aufgaben
- Kommunikation
- **Dokumenten**
- Daten
- Schwachstellen

Bestandteile

Dokumente, die verwendet und produziert werden

- Bezeichnung
  → *Kassenzettel*
- Inhalt
  → *gekaufte Ware, Preis, MWST, Datum, Verkäufer*
- Grad der Formalisierung, Aufbau
  → *genügt Gesetz*
- Verteiler
  → *Kunde und Laden*

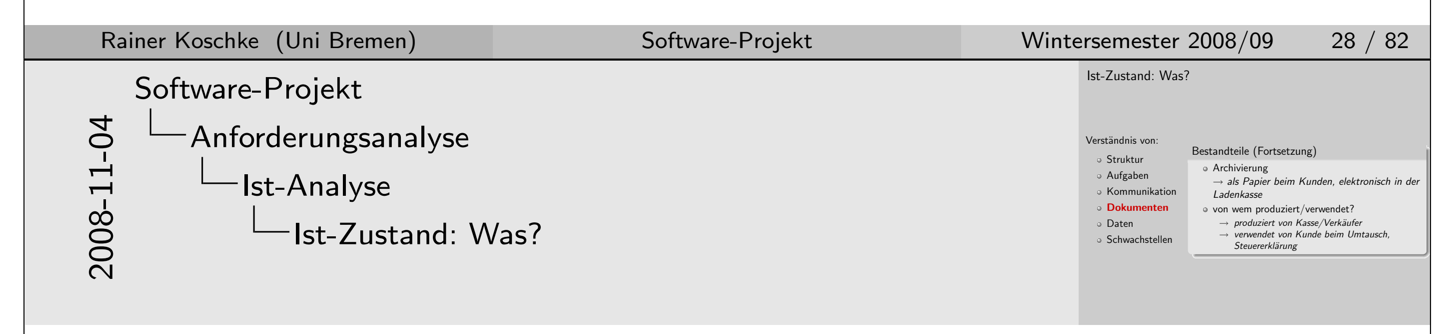
Ist-Zustand: Was?

Verständnis von:

- Struktur
- Aufgaben
- Kommunikation
- **Dokumenten**
- Daten
- Schwachstellen

Bestandteile (Fortsetzung)

- Archivierung
  → *als Papier beim Kunden, elektronisch in der Ladenkasse*
- von wem produziert/verwendet?
  → *produziert von Kasse/Verkäufer*
  → *verwendet von Kunde beim Umtausch, Steuererklärung*



Eine wichtige Informationsquelle bei der Ist-Analyse liefern existierende Dokumente, die verwendet und produziert werden. Meist müssen sie bei einer softwaretechnischen Lösung im Rechner nachgebildet werden. Dokumente können eher formalisiert sein, wie z.B. der Kassenbons im Fahrradladen, oder eher informell sein, wie z.B. der Notizzettel des Käufers, auf dem er sich aufgeschrieben hat, was er einkaufen möchte und zu welchem Fahrrad in seinem Besitz der neue Artikel passen soll.

Bei diesen Dokumenten interessieren uns die Bezeichnung, der Inhalt, der Grad der Formalisierung, der Aufbau, der Verteiler (d.h. wer das Dokument erhält), die Archivierung des Dokuments sowie die Produzenten und Konsumenten des Dokuments.

Bsp.: *Im Fahrradladen gibt es beispielsweise den Kassenzettel. Sein Inhalt ist die gekaufte Warte, der Preis, die anrechenbare Mehrwertsteuer, das Kaufdatum sowie der Verkäufer. Der Kassenzettel muss dem Gesetz genügen und ist darum recht formalisiert. Den Kassenzettel erhält der Kunde, ein Durchschlag bleibt in der Kasse des Ladens. Sowohl der Kunde als auch der Laden archivieren den Kassenzettel – der Kunde um gegebenenfalls seine Garantieansprüche durchzusetzen, der Laden für das Finanzamt. Produziert wird der Kassenzettel von der Kasse des Ladens durch den Verkäufer und verwendet wird er vom Kunden im Falle eines Umtausches beziehungsweise vom Ladenbesitzer für die Steuererklärung.* □

Ist-Zustand: Was?

Verständnis von:

- Struktur
- Aufgaben
- Kommunikation
- Dokumenten
- **Daten**
- Schwachstellen

Bestandteile

Umfang und Art der verarbeiteten Daten

- Volumen und Wachstum
  $\longrightarrow$ *41 Flaschenhalter*
- Wertebereiche
  $\longrightarrow$ *Radius Halterung, Abstand Schrauben, versch. Farben etc.*
- Datenträger
  $\longrightarrow$ *Katalog in Papierform, Online-Katalog in HTML*
- Ordnungsstrukturen
  $\longrightarrow$ *Name, Hersteller, Typ*
- Verarbeitungshäufigkeit
  $\longrightarrow$ *wird alle drei Tage einmal verkauft*

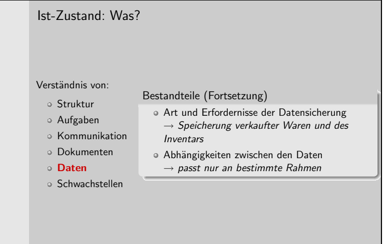
Ist-Zustand: Was?

Verständnis von:

- Struktur
- Aufgaben
- Kommunikation
- Dokumenten
- **Daten**
- Schwachstellen

Bestandteile (Fortsetzung)

- Art und Erfordernisse der Datensicherung
  $\longrightarrow$ *Speicherung verkaufter Waren und des Inventars*
- Abhängigkeiten zwischen den Daten
  $\longrightarrow$ *passt nur an bestimmte Rahmen*



Wir untersuchen die Dokumente, um die Daten zu ermitteln, die für die Ist-Analyse relevant sind. Oft sind es verschiedene Daten, die in den Dokumenten aufgeführt und verknüpft werden. Allerdings sind nicht alle relevanten Daten auf Dokumenten wiedergegeben. Im nächsten Schritt untersuchen wir weitere Daten, die möglicherweise nur mündlich ausgetauscht werden.

Um die Daten präzise und vollständig zu modellieren, wird hierzu oft ein Datenmodell aufgestellt. Das **Datenmodell** beschreibt die wesentlichen Daten, ihre Eigenschaften, ihren Umfang und ihre Beziehungen. Zur Repräsentation eignen sich für eine Reihe der relevanten Aspekte beispielsweise UML-Klassendiagramme. Das Datenmodell beschreibt das Volumen und zukünftig erwartete Wachstum der Daten. Dies wird häufig als das **Mengengerüst** bezeichnet. Es ist unerlässlich für die Abschätzung des Speicherbedarfs und der Anforderungen an die Performanz der Algorithmen. Mit der Betrachtung des zukünftigen Wachstums stellen wir sicher, dass unser System skaliert und auch in einigen Jahren noch die notwendige Performanz zur Verfügung stellen wird. Die Daten haben einen Typ, für den wir den Wertebereich angeben. Sie werden in vielen Fällen dauerhaft gespeichert auf Datenträgern, die aber nicht notwendigerweise unmittelbar elektronisch verarbeitbar sind, z.B. weil sie von Hand auf Papier geschrieben werden. Viele Daten können sortiert werden, d.h. verfügen über ein oder mehrere Ordnungskriterien. Unsere Programme müssen sie meist nach diesen Ordnungskriterien sortieren können. Die Häufigkeit ihrer Verarbeitung ist für uns wichtig, weil wir damit die Zugriffe entsprechend organisieren müssen. Häufig verarbeitete Daten müssen schnell verfügbar sein. Schließlich interessieren uns die Art und Erfordernisse der Datensicherung sowie die Abhängigkeiten zwischen den Daten. Häufige solche Abhängigkeiten sind die Teil-Von-Beziehung in Form der Aggregation und Komposition. Andere Beziehungen werden in UML-Klassendiagrammen mit allgemeinen Assoziationen modelliert.


2008-11-04
Software-Projekt
└─Anforderungsanalyse
└─Ist-Analyse
└─Ist-Zustand: Was?
Ist-Zustand: Was?
Verständnis von:
Struktur
Aufgaben
Kommunikation
Dokumenten
Daten
Schwachstellen
Bestandteile (Fortsetzung)
Art und Erfordernisse der Datensicherung
→ Speicherung verkaufter Waren und des Inventars
Abhängigkeiten zwischen den Daten
→ passt nur an bestimmte Rahmen

Bsp.: *In unserem Fahrradladen gibt es beispielsweise 41 Flaschenhalter in fünf verschiedenen Typen von drei verschiedenen Herstellern. Ein Flaschenhalter wird beschrieben durch die Attribute* $Radius\ der\ Halterung$ *gemessen in Zentimetern, der Abstand der Schrauben gemessen in Zentimetern, die Farben in rot, grün, gelb, blau sowie der Preis im Bereich von 5 bis 25 Euro. Die Datenträger zu den Flaschenhaltern sind die Herstellerkataloge in Papierform als auch als Online-Kataloge in HTML. Name, Hersteller, Typ und Preis bilden die Ordnungsstrukturen. Von den Flaschenhaltern wird im Durchschnitt alle drei Tage einer verkauft. Dies bildet die Verarbeitungshäufigkeit. Die verkauften und im Lager befindlichen Flaschenhalter werden in einer Datenbank des Ladenbesitzers gespeichert. Die Flaschenhalter passen nur an bestimmte Rahmen; dies hängt vom Schraubenabstand und dem Radius ab. Eine weitere „weiche“ Abhängigkeit ist die Farbe des Rahmens, zu der der Flaschenhalter passen soll.* □

Ist-Zustand: Was?

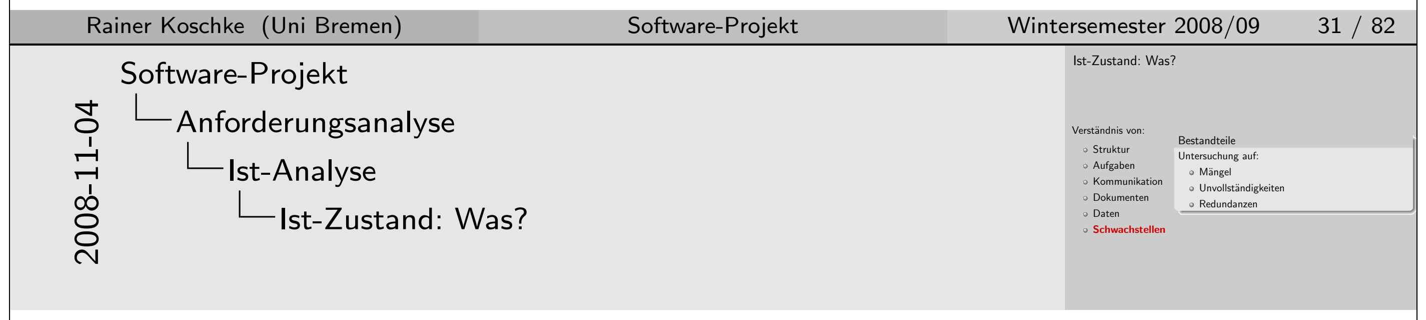
Verständnis von:

- Struktur
- Aufgaben
- Kommunikation
- Dokumenten
- Daten
- **Schwachstellen**

Bestandteile

Untersuchung auf:

- Mängel
- Unvollständigkeiten
- Redundanzen



Nachdem wir die oben genannten relevanten Aspekte des Ist-Zustands erfasst haben, analysieren wir sie auf Schwachstellen und Probleme. Dazu gehören Mängel, Unvollständigkeiten oder Redundanzen. Auf die Schwachstellen werden wir sowohl durch eigene Analyse und Beobachtung der erhobenen Aspekte als auch durch Befragung und Beobachtung der Akteure selbst aufmerksam. Zu entsprechenden Techniken der Befragung und Beobachtung kommen wir weiter unten.

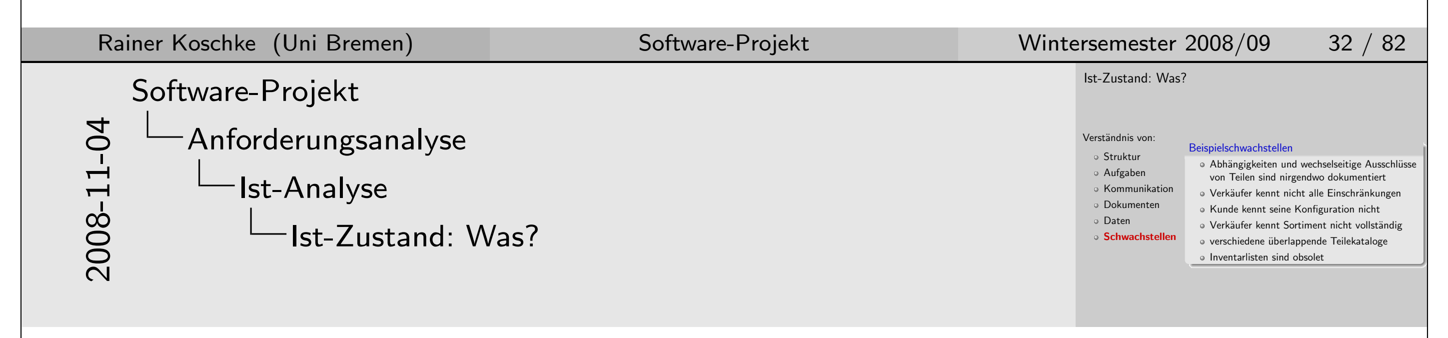
Ist-Zustand: Was?

Verständnis von:

- Struktur
- Aufgaben
- Kommunikation
- Dokumenten
- Daten
- **Schwachstellen**

Beispielschwachstellen

- Abhängigkeiten und wechselseitige Ausschlüsse von Teilen sind nirgendwo dokumentiert
- Verkäufer kennt nicht alle Einschränkungen
- Kunde kennt seine Konfiguration nicht
- Verkäufer kennt Sortiment nicht vollständig
- verschiedene überlappende Teilekataloge
- Inventarlisten sind obsolet

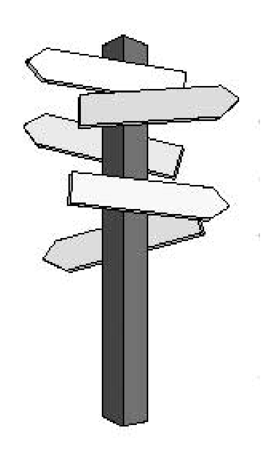


Bsp.: *In unserem Fahrradladen können wir beispielsweise durch unsere Analyse der erhobenen Aspekte und durch Beobachtung im Laden folgende Schwachstellen identifizieren:*

- *Die Abhängigkeiten und wechselseitige Ausschlüsse der Bestandteile eines Fahrrads sind nirgendwo dokumentiert. Ein Verkäufer kann sie nicht nachschlagen und muss sie alle kennen, um einen Kunden zu beraten.*
- *Nicht alle Verkäufer kennen aber alle Einschränkungen. Dies führt dazu, dass weniger erfahrene Verkäufer Rücksprach halten müssen mit erfahreneren Verkäufern oder gar Lieferanten und dass Kunden verärgert ihre falschen Artikel umtauschen müssen, wenn sie falsch beraten wurden.*
- *Die Verkäufer können die Kunden nicht ausreichend beraten, weil der Kunde die bereits gekauften Fahrradbestandteil nicht genau kennt, zu denen der neu zu kaufende Artikel passen soll. Der Kunde muss in solchen Fällen unverrichteter Dinge den Laden wieder verlassen und die Information besorgen.*
- *Verkäufer kennen das Sortiment nicht vollständig und wissen nicht genau, was im Laden selbst oder im Lager noch verfügbar ist.*
- *Die Fahrradartikel sind in verschiedenen teilweise überlappenden Teilekatalogen beschrieben. Dies führt zu unnötigen Redundanzen.*
- *Die Inventarlisten sind obsolet. Der Ladenbesitzer weiß nicht genau, welche Ware noch vorrätig, verkauft oder gar gestohlen ist.*

□

Wegweiser



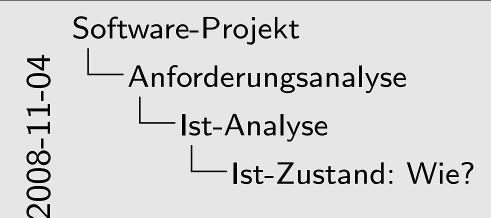
Ist-Analyse

Welchen Techniken können wir in der Ist-Analyse nutzen?

Ist-Zustand: Wie?

Erhebungstechniken

- Auswertung vorhandener Dokumente
- Befragung
  - schriftlicher Fragebogen
  - Interview
- Beobachtung
  - anekdotisch ↔ systematisch
  - teilnehmend ↔ nicht-teilnehmend
  - offen ↔ verdeckt
  - selbst ↔ fremd
  - Feld ↔ Labor



Erhebungstechniken

- Auswertung vorhandener Dokumente
- Befragung
  - schriftlicher Fragebogen
  - Interview
- Beobachtung
  - anekdotisch ↔ systematisch
  - teilnehmend ↔ nicht-teilnehmend
  - offen ↔ verdeckt
  - selbst ↔ fremd
  - Feld ↔ Labor

Wir wissen nun, was wir erheben sollen, aber leider noch nicht, wie wir das tun können. In diesem Abschnitt lernen wir hierfür verschiedene Erhebungstechniken kennen. Die Auswertung der vorhandenen Dokumente ist eine erste offensichtliche Technik. Darüber hinaus können wir jedoch auch die Akteure fragen oder beobachten.

Um Akteure zu befragen, können wir sie selbst interviewen oder ihnen aber einen schriftlichen Fragebogen zuschicken. Beim Interview kann man auch mit einem vorher gründlich ausgearbeiteten Fragebogen arbeiten oder aber nur eine grobe Skizze der Fragen haben, um dann die Fragen an die Situation spontan anzupassen. Ganz zu Anfang, wenn man sich ein Gebiet erschließen möchte, ist man oft gar nicht in der Lage, präzise Fragen auszuarbeiten. Das Interview hat gegenüber der Befragung durch einen schriftlichen Fragebogen, den der Befragte ohne unser Beisein ausfüllt, den Vorteil, dass wir auch Gestik und Mimik des Befragten sehen, seine Gegenfragen direkt klären können und individuell auf die Situation reagieren können. Allerdings kann die Anwesenheit eines Fragenden die Antworten des Befragten beeinflussen. Zum einen lässt sich die Anonymität hier nicht sicher stellen, zum anderen fallen die Antworten oft auch abhängig davon aus, inwieweit der Fragende dem Befragten sympathisch ist. Außerdem ist das individuelle Interview erheblich zeitaufwändiger und teurer. Bei den schriftlichen Fragebögen verhält es sich gerade umgekehrt. Aus diesem Grunde werden wir meistens sowohl Interviews als auch schriftliche Fragebögen anwenden, weil sie sich bestens ergänzen. Wir werden meist zu Anfang eine kleinere Zahl von Interviews führen. Später können wir auf Basis dieser Interviews präzise Fragebögen aufstellen, um eine größere Datenbasis zu schaffen.

2008-11-04



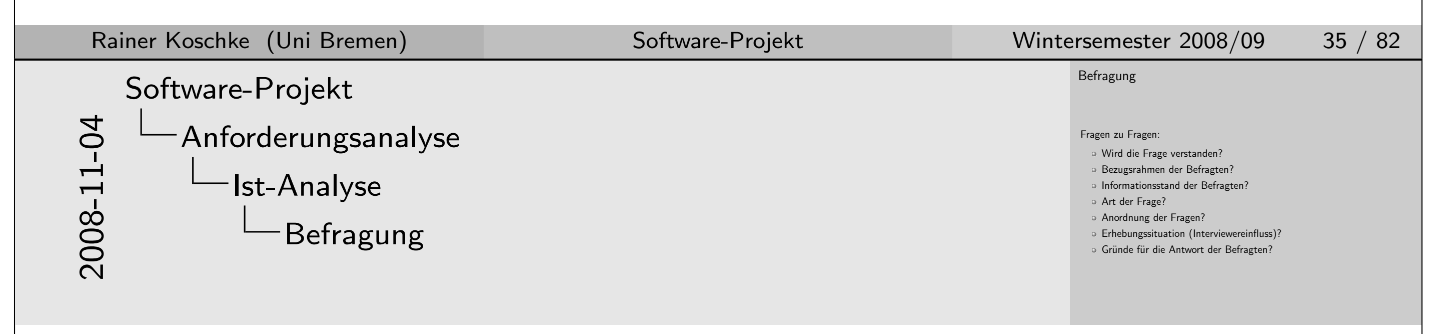
Statt zu fragen können wir auch beobachten. Beobachtungen haben aber viele Facetten, aus denen wir die geeignete Beobachtungsart zusammen stellen müssen. Hier müssen wir zunächst entscheiden, ob wir systematisch oder eher anekdotisch vorgehen wollen. Wenn wir z.B. ganz einfach einmal den Fahrradladen betreten und dort eine Stunde verbringen, liefert das eher anekdotisches Wissen. Bei einem systematischen Vorgehen würden wir vorher ermitteln, wie sich die Kundenbesuche über die Ladenöffnungszeiten verteilen. Dann würden wir zu qualitativ unterschiedlichen Zeiten den Laden betreten, z.B. Montag morgens, wenn nicht viel los ist, Mittwoch abends, wenn mehr los ist, und Samstag vormittags, wenn der Laden meist voll ist. Außerdem müssen wir entscheiden, ob wir bei der Beobachtung teilnehmen. Wir könnten uns also einerseits nur einfach in den Laden stellen und beobachten, oder aber könnten wir selbst Kunde spielen und uns beraten lassen. Die Beobachtung kann offen oder verdeckt erfolgen. Erfolgt sie offen, kann das den Beobachtungsgegenstand beeinflussen. Ein Verkäufer wird sich in einer solchen Situation zum Beispiel eher freundlich verhalten, als wenn er sich unbeobachtet fühlt. Andererseits werfen verdeckte Beobachtungen ethische Fragen auf und sind möglicherweise schlicht verboten. Dann müssen wir uns entscheiden, ob wir selbst beobachten oder die Beobachtung Dritten überlassen. Wenn wir selbst beobachten, haben wir ein unmittelbareres Bild. Andererseits sind wir möglicherweise ungeübt, und ausgebildete und erfahrene dritte Beobachter sehen mehr als wir. Schließlich müssen wir uns zwischen einer Feld- oder Laborbeobachtung entscheiden. Im Labor können wir viel stärker beeinflussende Faktoren kontrollieren, um so spezielle Situationen und den Einfluss der Faktoren durch deren Variation genauer zu studieren. So könnten wir z.B. eine stressige Ladensituation im Labor nachbilden, indem wir sehr viele Menschen unzufriedene Kunden schauspielern lassen. Das natürlichere Bild ergibt sich aber durch die Feldbeobachtung. Hier lassen sich aber keine Situationen nachstellen.

Auch hier gilt wieder, dass man Für und Wider anhand der konkreten Ausgangslage und des Ziels abwägen muss. Eine Feldbeobachtung ist jedoch immer vernünftig. Im Labor kann man dann bestimmte Situationen nachstellen und näher untersuchen.

Befragung

Fragen zu Fragen:

- Wird die Frage verstanden?
- Bezugsrahmen der Befragten?
- Informationsstand der Befragten?
- Art der Frage?
- Anordnung der Fragen?
- Erhebungssituation (Interviewereinfluss)?
- Gründe für die Antwort der Befragten?



Die Antwort auf eine Frage hängt nicht nur vom Inhalt der Frage selbst, sondern auch vom Kontext und von der Art und Weise ab, wie man die Frage stellt. Aus diesem Grunde sollte man sich bei der Gestaltung eines Fragebogens die Fragen und die Art der Frage sorgfältig überlegen und sich über den Einfluss des Kontextes im Klaren werden. Die Sozialwissenschaften haben hierzu umfangreiche Lehrbücher entwickelt.

Zu unseren Fragen müssen wir uns selbst folgende Fragen stellen:

- Wird die Frage überhaupt richtig verstanden? Benutzen wir Begriffe, zu denen der Befragte die gleichen Vorstellungen hat wie der Fragende? Bsp.: *Wenn wir Softwaretechniker von Server sprechen und damit einen Prozess meinen, dann versteht der Befragte darunter möglicherweise den Rechner, auf dem seine Applikation laufen, wenn er etwas Hintergrund in Computer-Hardware hat, oder gar lediglich $Diener$, wenn er Computer-Laie ist.*
- Was ist der Bezugsrahmen der Befragten? Was ist seine Rolle, was kann er wissen, was sind seine Ziele sowohl beruflicher als auch persönlicher Art? Bsp.: *Es hat keinen Sinn den Besitzer unseres Fahrradladens zum üblichen Abla von Verkaufsgesprächen zu fragen, wenn er selbst seit langer Zeit das Verkaufen nicht mehr selbst praktiziert hat und lediglich Managementaufgaben wahrnimmt.*
- Informationsstand der Befragten? Was kann der Befragte überhaupt wissen? Fragen wir ihn zu Dingen, die er nicht wissen kann, wird er keine Antwort abgeben können oder schlimmer noch, eine Antwort erfinden.
- Art der Frage? Ist eine offene, geschlossene oder hybride Frage angebracht (siehe unten)?
- Anordnung der Fragen? Die Reihenfolge der Fragen kann das Ergebnis beeinflussen. Über jede Frage denkt der Befrag nach. Die Assoziationen können einen Kontext herstellen, der die Antwort auf die nächste Frage beeinflusst. Bsp.: *Um ein krasses Beispiel zu nennen: Wenn wir den Verkäufer in unserem Fahrradladen als erstes fragen würden, ob er sich durch den geplanten Einsatz unseres Softwaresystems unnütz fühlen würde, wird das vermutlich Einfluss auf den Verla der weiteren Befragung nehmen.*

Befragung

Fragen zu Fragen:
- Wird die Frage verstanden?
- Bezugsrahmen der Befragten?
- Informationsstand der Befragten?
- Art der Frage?
- Anordnung der Fragen?
- Erhebungssituation (Interviewereinfluss)?
- Gründe für die Antwort der Befragten?

- Was ist der mögliche Einfluss der Situation, in der der Fragebogen erhoben wird? Was ist insbesondere der Einfluss de Interviewers, falls die Fragen in einem Interview gestellt werden? Bsp.: *Fragt man Menschen, die eben aus einem Kinofilm zu einer Naturkatastrophe – wie z.B. einer Flutwelle – kommen, zum Thema Naturschutz, sind andere Antworten zu erwarten, wie wenn wir die selben Menschen bei einem Transatlantikflug zu ihrem Urlaubsort fragen.* Auch der Fragende hat einen möglichen Einfluss auf die Antworten. Ob uns jemand sympathisch oder unsympathisch i beeinflusst, wie kooperativ wir in einem Interview sind. Fatal wäre es, wenn der Befragte vor dem Fragenden etwas zu befürchten hätte. Bsp.: *So wäre der Ladenbesitzer eher ungeeignet, Fragen zum Thema Auslastung und Schwächen d Verkäufer zu stellen, wenn sie zu befürchten haben, dass ihnen aus ehrlichen Antworten negative Konsequenzen erwachsen könnten.*
- Was sind die möglichen Gründe für die Antwort der Befragten? Der Befragte könnte uns einen Gefallen tun wollen od will uns sabotieren. Er könnte bewusst oder unbewusst falsche Antworten liefern, weil er hinter den Fragen ein bestimmtes Ziel vermutet, mit dem er nicht einverstanden ist.

Fragetypen: Geschlossene Fragen

Geschlossene Fragen:

Welche Qualität hat die GUI? Bitte ankreuzen.

☐ *sehr gut*
☐ *gut*
☐ *schlecht*
☐ *weiß nicht*

- Antwortalternativen vorgegeben
- auch Mehrfachantworten

Fragetypen: Geschlossene Fragen

Geschlossene Fragen:

Welche Qualität hat die GUI? Bitte ankreuzen.

- ☐ *sehr gut*
- ☐ *gut*
- ☐ *schlecht*
- ☐ *weiß nicht*

- Antwortalternativen vorgegeben
- auch Mehrfachantworten

Fragen können in drei unterschiedlichen Formen verfasst sein: geschlossene, offene oder hybride Fragen. Sie unterscheiden sich durch die Form der Antworten, die auf sie möglich sind.

Bei geschlossenen Fragen sind alternative Antworten bereits vorgegeben, aus denen der Befragte eine oder gegebenenfalls mehrere auswählen kann. Häufig findet man solche Fragen für Aspekte, die anhand einer Skala eingeschätzt werden sollen, also zum Beispiel die Frage, ob einem die GUI sehr gut, gut oder nicht gefällt.

Durch die Vorgabe möglicher Antworten schafft man eine Normierung und kann die Antworten leicht automatisiert statistisch auswerten.

Diesen Fragentyp kann man gut einsetzen, wenn man die möglichen Antworten gut vorwegsehen kann. Wenn man also bereits eine bestimmte Hypothese hat, kann man sie durch solche Fragen leicht überprüfen. In Bereichen, in denen man jedoch die Antworten nur schlecht abschätzen kann, besteht die Gefahr, dass die Antwort, die jemand eigentlich geben möchte, nicht dabei ist. Hier sind solche Fragetypen eher ungeeignet. In jedem Falle sollte es bei der Antwort immer möglich sein, dass der Befragte die Frage nicht beantworten kann oder will, um erzwungene Falschantworten zu vermeiden.

Fragetypen: Offene Fragen

Offene Fragen:

Wie sollte die GUI verbessert werden?

- Antworten in eigenen Worten, im eigenen Referenzsystem
- erfordert Ausdrucksfähigkeit der Befragten
- starker Einfluss des Fragenden, wenn präsent (durch Aufschreiben, Weglassen)
- hoher Auswertungsaufwand

Fragetypen: Offene Fragen

Offene Fragen:
Wie sollte die GUI verbessert werden?

- Antworten in eigenen Worten, im eigenen Referenzsystem
- erfordert Ausdrucksfähigkeit der Befragten
- starker Einfluss des Fragenden, wenn präsent (durch Aufschreiben, Weglassen)
- hoher Auswertungsaufwand

Offene Fragen sind solche, bei denen keine Antwort vorgegeben wird. Der Befragte beantwortet die Frage mit seinen eigenen Worten. Dazu kann er seine eigene Terminologie benutzen. Dies erfordert vom Befragten, dass er sich selbst ausdrücken kann. Da jede Antwort möglich ist, kann der Befragte in seiner Formulierung stark durch den anwesenden Fragenden beeinflusst werden. Dies gilt umso mehr, wenn nicht der Befragte die Antworten aufschreibt, sondern der Fragende, wie das meist üblich ist. Dadurch werden die Aussagen vom Fragenden oft gefiltert und aufbereitet, statt nur wörtlich wiedergegeben. Das Mindeste, was man dann tun sollte, ist es, dem Befragten das Protokoll seiner Antworten nochmals zur Kontrolle vorzulegen, um sicher zu stellen, dass man seine Aussagen korrekt wieder gegeben hat.

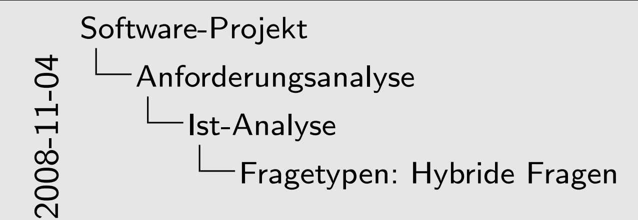
Da die Antworten nicht vorgegeben sind, muss ein Mensch sie durchlesen, verstehen und einordnen. Damit ist die Auswertung erheblich aufwändiger als bei geschlossenen Fragen. Dem hohen Auswerteaufwand steht als Vorteil gegenüber, dass man die Antworten nicht schon beim Entwurf des Fragebogens vorwegsehen muss, wie das bei geschlossenen Fragen der Falle ist. Man hat also die Chance, Neues zu erfahren. Aus diesem Grund wendet man diesen Fragetyp an, wenn man noch keine rechte Hypothese zu den möglichen Antworten hat.

Fragetypen: Hybride Fragen

Hybride Fragen:

Was stört Sie an der GUI?
☐ *lange Reaktionszeit*
☐ *mangelnde Selbsterklärungsfähigkeit*
☐ *fehlendes „Undo“*
☐ *umständliche Dialogführung*
☐ ________________________________

- Kombination von geschlossenen und offenen Fragen



Hybride Fragen:

Was stört Sie an der GUI?

- ☐ *lange Reaktionszeit*
- ☐ *mangelnde Selbsterklärungsfähigkeit*
- ☐ *fehlendes „Undo"*
- ☐ *umständliche Dialogführung*
- ☐ ____________

- Kombination von geschlossenen und offenen Fragen

Hybride Antworten kombinieren geschlossene und offene Fragen, indem eine Menge alternativer Antworten vorgegeben wird, es aber dennoch möglich ist, durch Freitextfelder eigene Antworten zu formulieren. Dies ist in vielen Fällen ein guter Kompromiss, der leichte Auswertbarkeit und die Möglichkeit, unvorhergesehene Antworten zu liefern, bestmöglich vereint, solange die vorgegebenen Antworten möglichst viele echte Antworten abdecken.

Wann welche Erhebungsform?

weniger strukturiertes Interview

- unstrukturiertes Untersuchungsgebiet
- offene Gesprächsführung und größere Antwortspielräume
- persönlicher Kontakt möglich
- Ortsbegehung möglich

stark strukturierter Fragebogen

- vorstrukturiertes Untersuchungsgebiet
- gute Kenntnisse des Untersuchungsgebiets
- Operationalisierung der Hypothesen möglich

Wann welche Erhebungsform?

weniger strukturiertes Interview
- unstrukturiertes Untersuchungsgebiet
- offene Gesprächsführung und größere Antwortspielräume
- persönlicher Kontakt möglich
- Ortsbegehung möglich

stark strukturierter Fragebogen
- vorstrukturiertes Untersuchungsgebiet
- gute Kenntnisse des Untersuchungsgebiets
- Operationalisierung der Hypothesen möglich

Welche Form der Erhebung, ob stark oder weniger strukturiertes Vorgehen, nun angemessen ist, hängt im Wesentlichen davon ab, wie gut wir unser Untersuchungsgebiet bereits durchdrungen haben.

In einer frühen Phase, in der wir uns noch wenig auskennen und mögliche Antworten nicht vorweg sehen können, werden wir eher ein weniger strukturiertes Interview durchführen. Das Gespräch kann offen geführt werden, und der Befragte hat einen größeren Spielraum bei Antworten. Der Fragende kann spontaner auf die Situation reagieren. Der persönliche Kontakt und die Ortsbegehung werden uns weitere Einblicke gewähren.

Ist unser Untersuchungsgebiet bereits gut vorstrukturiert, dann kommt auch ein strukturierteres Vorgehen in Frage. Da wir bereits über gute Kenntnisse des Untersuchungsgebiets verfügen, sind wir in der Lage, operationale Hypothesen aufzustellen, das heißt, Hypothesen, die wir objektiv durch Fragebögen überprüfen können.

Frageformulierung

- einfache Worte
- kurz und konkret
- keine doppelten Negationen
- nur auf einen Sachverhalt bezogen
- nicht hypothetisch
- neutral, nicht suggestiv
- Befragten nicht überfordern
- balanciert (negative und positive Antwortmöglichkeiten)
- immer eine „weiß-nicht“-Kategorie bieten

Frageformulierung

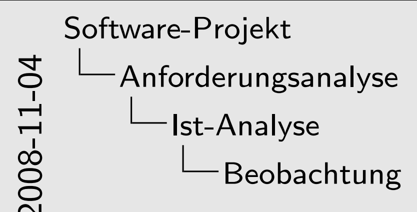
- einfache Worte
- kurz und konkret
- keine doppelten Negationen
- nur auf einen Sachverhalt bezogen
- nicht hypothetisch
- neutral, nicht suggestiv
- Befragten nicht überfordern
- balanciert (negative und positive Antwortmöglichkeiten)
- immer eine „weiß-nicht"-Kategorie bieten

Auch über die Formulierung der Fragen muss man sich vorher sorgfältig Gedanken machen, da sie einen Einfluss auf die Antwort hat. Man wähle möglichst einfache, klare Worte, so dass die Frage sicher verstanden werden kann. Die Fragen sollten kurz und konkret sein. Lange Fragen beinhalten viele Aspekte, auf die man kaum eine klare, abgeschlossene Antwort geben kann. Auf abstrakte Fragen fällt die Antwort schwer. Doppelt negierte Fragen gilt es zu vermeiden, um den Befragten nicht zu verwirren. Gleichermaßen sollte man auf hypothetische Fragen mit vielen Konjunktiven verzichten. Ansonsten kann eine negative Antwort unterschiedliche Ursachen haben: die, die sich auf den eigentlichen Frageinhalt beziehen, oder jene, die die Prämissen in Frage stellen. Die Frage sollte neutral und nicht suggestiv gestellt werden, ansonsten legt man dem Befragten die Antwort bereits in den Mund. Man sollte eine gleiche Anzahl negativer und positiver Antwortmöglichkeiten vorsehen. Eine Disbalance von entweder negativen oder positiven Fragen wirkt suggestiv. Außerdem gibt es eine menschliche Tendenz, eher positiv zu antworten. Auch bei geschlossenen Fragen sollte es stets möglich sein, dass der Befragte aussagen kann, dass er auf eine Frage nicht antworten kann oder will, um erzwungene Falschaussagen zu vermeiden.

Beobachtung

Prinzipien der Ethnographie

- Natürliche Umgebung
  → Aktivitäten in Alltagsumgebung untersuchen
- Ganzheitlichkeit
  → Einzelverhalten im Kontext untersuchen
- Beschreiben, nicht bewerten
  → Ist-Verhalten, nicht Soll-Verhalten
- Sicht der Handelnden einnehmen
  → Verhalten beschreiben in Begriffen, die für den Handelnden relevant und bedeutungsvoll sind



Prinzipien der Ethnographie

- Natürliche Umgebung
  → Aktivitäten in Alltagsumgebung untersuchen
- Ganzheitlichkeit
  → Einzelverhalten im Kontext untersuchen
- Beschreiben, nicht bewerten
  → Ist-Verhalten, nicht Soll-Verhalten
- Sicht der Handelnden einnehmen
  → Verhalten beschreiben in Begriffen, die für den Handelnden relevant und bedeutungsvoll sind

In der Befragung adressieren wir hauptsächlich das bewusste Wissen. Wir haben weiter oben schon gelernt, dass das nur rund ein Drittel des menschlichen Wissens ausmacht. Durch Beobachtung können wir uns noch weiteres Wissen erschließen.

In der Sozialwissenschaft hat sich die Ethnographie als Technik etabliert. Die Ethnographie ist eine teilnehmende Beobachtung. Es hat offensichtlich keinen Sinn, das Leben von Urvölker zu ergründen, indem man sie ihrer Heimat entreißt und sie im Labor untersucht. Eigenschaften von Sozialgemeinschaften sind abhängig vom Kontext und müssen deshalb in diesem Kontext untersucht werden.

Die Ethnographie folgt bei der Beobachtung einer Reihe von Prinzipien:

- Die soziale Gemeinschaft wird in ihrer natürlichen Umgebung beobachtet. Auf diese Weise werden ihre Aktivitäten in d Alltagsumgebung untersucht.
- Das Einzelverhalten wird im Kontext untersucht. Verhaltensweisen sind meist sehr komplex und lassen sich nur schwer isolieren und kontrollieren, wie das für Laborversuche versucht wird.
- Im Vordergrund steht das neutrale Beschreiben und nicht die Bewertung. Wertung sozialen Handelns orientiert sich an moralischen Maßstäben, die jedoch kontextabhängig sind. Setzen wir uns die eigene moralische Brille auf, werden wir meist schlechtsichtiger und übersehen wichtige Details. Wir sind bei der Ist-Analyse am Ist- und nicht am Soll-Verhalt interessiert.
- Wir versuchen, die Sicht der Handelnden einzunehmen, d.h. wir beschreiben das Verhalten in Begriffen, die für den Handelnden relevant und bedeutungsvoll sind.

Die Einhaltung dieser Prinzipien sichert uns einen besseren Einblick.

Interview im Kontext

- ist eine Form der ethnographischen Untersuchung
- nach dem Meister-Lehrling-Modell
- Lernen durch Vormachen und Beobachten sowie Fragen und Klären
- geprägt durch
  - Bescheidenheit
  - Neugier
  - Aufmerksamkeit
  - konkrete (statt abstrakter) Fragen

Interview im Kontext

- ist eine Form der ethnographischen Untersuchung
- nach dem Meister-Lehrling-Modell
- Lernen durch Vormachen und Beobachten sowie Fragen und Klären
- geprägt durch
  - Bescheidenheit
  - Neugier
  - Aufmerksamkeit
  - konkrete (statt abstrakter) Fragen

Eine spezielle Form der ethnographischen Untersuchung ist das Interview im Kontext. Es orientiert sich am Meister-Lehrling-Modell. Lernen findet hier in der Beobachtung des Meisters, im aktiven Nachfragen und Zeigen sowie im Mitmachen statt. Statt in einem Frontalunterricht im Hörsaal zu theoretisieren, nimmt der Meister seinen Lehrling mit auf die Baustelle. Der Meister arbeitet vor und der Lehrling lernt aus der Beobachtung. Der Lehrling fragt nach, wenn ihm etwas nicht klar ist, und er versucht sich auch selbst. Das Meister-Lehrling-Modell ist geprägt durch Bescheidenheit, Neugier, Aufmerksamkeit und konkrete (statt abstrakter) Fragen des Lehrlings im unmittelbaren Kontext.

Beispiel eines Ablaufes

1. Einleitung (15 min)
   - Vorstellung, Ziele, Dank
   - Zustimmung zu Aufzeichnung, Vertraulichkeit
   - Arbeit, nicht Person wird betrachtet!
   - Meinungen zu technischer Unterstützung?
   - Überblick gewinnen
2. Übergang (1 min)
   - Regeln, Rollen, Beziehung
   - ich frage, Sie dürfen abwehren
3. Erhebung im Kontext (2 Std.)
   - Beobachtung und Nachfragen
   - Notizen machen, mitlaufen, sich unsichtbar machen
   - Pausen nach Wunsch
4. Zusammenfassung (15 min)
   - was die Beschäftigte tut, ihre Rolle
   - was wichtig ist
   - Ergänzungen, Korrekturen?

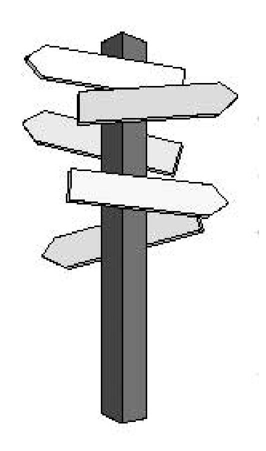
Beispiel eines Ablaufes

1. Einleitung (15 min)
   - Vorstellung, Ziele, Dank
   - Zustimmung zu Aufzeichnung, Vertraulichkeit
   - Arbeit, nicht Person wird betrachtet!
   - Meinungen zu technischer Unterstützung?
   - Überblick gewinnen
2. Übergang (1 min)
   - Regeln, Rollen, Beziehung
   - ich frage, Sie dürfen abwehren
3. Erhebung im Kontext (2 Std.)
   - Beobachtung und Nachfragen
   - Notizen machen, mitlaufen, sich unsichtbar machen
   - Pausen nach Wunsch
4. Zusammenfassung (15 min)
   - was die Beschäftigte tut, ihre Rolle
   - was wichtig ist
   - Ergänzungen, Korrekturen?

Ein Interview im Kontext dauert in der Regel zwei Stunden plus direkte Vor- und Nachbearbeitung. Es könnte wie folgt ablaufen:

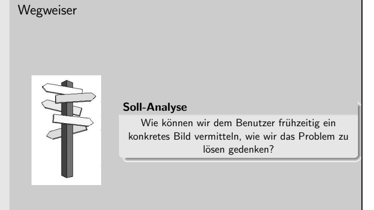
1. In einer **Einleitung** (15 min) stellt sich der Fragende kurz vor und erläutert das Ziel der Befragung. Er vergisst nicht, sich ausdrücklich für das Interview zu bedanken. Er sichert absolute Vertraulichkeit zu und vergewissert sich über das Einverständnis des Befragten. Er weist darauf hin, dass die Arbeit betrachtet werden soll und nicht die Person. Man kann technische Hilfen für das Interview mitbringen, wie z.B. ein Audio- oder Videoaufnahmegerät. Wenn das der Fall ist, holt man sich spätestens jetzt noch das Einverständnis des Beobachteten ein.
2. Im **Übergang** zur Erhebung im Kontext (1 min) bespricht man die Regeln, Rollen und die Beziehung zwischen Beobachter und Beobachtetem. Die Rolle des Beobachters ist es, zu beobachten und Fragen zu stellen. Die Rolle des Beobachteten ist es, sich möglichst wie gewöhnlich zu verhalten. Eine Grundregel lautet, dass der Beobachtete Fragen auch abwehren darf.
3. Der Hauptteil ist die **Erhebung im Kontext**, die zwei Stunden möglichst nicht überschreiten sollte, weil spätestens dar die Konzentration auf beiden Seiten nachgelassen hat. In der Erhebung beobachtet man und fragt nach. Man macht sich Notizen, läuft mit, macht sich aber möglichst unsichtbar, um das beobachtete Geschehen so wenig wie möglich z beeinflussen. Pausen können nach Wunsch gemacht werden.
4. In der **Zusammenfassung** (15 min), erläutert man, was man beobachtet hat und lässt den Beobachteten korrigieren u ergänzen. Der Beobachtete erläutert sein Tun und seine Rolle und weist auf Wichtiges hin.

Wegweiser



Soll-Analyse

Wie können wir dem Benutzer frühzeitig ein konkretes Bild vermitteln, wie wir das Problem zu lösen gedenken?



Durch Analyse von Dokumenten, Beobachtungen und Fragen haben wir den Ist-Zustand erfasst. Die Analyse des Erfassten deckt die Schwachstellen auf, die wir mit einer softwaretechnischen Lösung beseitigen sollen. Der nächste Schritt ist nun auszuarbeiten, was die softwaretechnische Lösung dazu leisten muss. Dieser Aufgabe widmet sich die Soll-Analyse. Die Soll-Analyse muss die Anforderungen an die zu implementierende Software aufstellen. Dazu gehört es auch, diese Anforderungen hinsichtlich ihrer Machbarkeit und Folgen einzuschätzen. In diesem Abschnitt lernen wir eine Technik kennen, mit der wir sowohl Anforderungen überprüfen als auch technische Machbarkeit untersuchen können.

Beobachtungen setzen den beobachteten Gegenstand voraus. Somit zielen Beobachtungen klar auf den Ist-Zustand ab. Unsere softwaretechnische Lösung existiert aber noch nicht. Fragen hingegen sind sowohl hilfreiche Mittel für die Ist-Analyse als auch für die Soll-Analyse. Wir können z.B. die Benutzer fragen, wie etwas sein soll. Auf welche Probleme wir dabei allerdings stoßen, haben wir schon ganz zu Anfang bei der Einführung zur Anforderungsanalyse besprochen. Die Benutzer können oft auf Problem hinweisen, sie tun sich oft aber schwerer, eine geeignete Lösung zu ersinnen. Insbesondere fehlt ihnen dazu meist die realistische Einschätzung zur technischen Machbarkeit, für die sie einen softwaretechnischen Hintergrund bräuchten. Zudem richten sich Fragen an das bewusste Wissen und wir können damit nur eingeschränkt relevante Aspekte erfassen.

Wenn aber Beobachtungen ausscheiden und Fragen nicht hinreichend sind, wie können wir dann vorgehen? Es gibt das geflügelte Wort des Benutzers: *I know it when I see it*. Ich weiß, was ich will, sobald ich es sehe. Dieses Prinzip können wir umsetzen: Wir führen dem Benutzer unsere Lösung vor, zu der er dann Stellung nehmen kann. Natürlich können wir uns es nicht leisten, gleich das gesamte System zu implementieren, denn der Schaden wäre groß, wenn diese Lösung dann doch nicht das war, was der Benutzer haben möchte. Statt dessen implementieren wir einen so genannten Prototypen.

Prototyping

Zielsetzung:

- Anforderungen anhand eines Beispiels erheben und überprüfen
- technische Möglichkeiten überprüfen und demonstrieren
- frühzeitig mögliche Lösungsansätze präsentieren

Idee:

- rasche und billige Entwicklung eines prototypischen Systems als Diskussionsgrundlage

Typen unterscheiden sich in ...

- Lebensdauer: Wegwerfprototyp, evolutionärer Prototyp
- Zweck: technische Machbarkeit, Demonstration der Funktionalität oder Interaktion

Prototyping

Zielsetzung:
- Anforderungen anhand eines Beispiels erheben und überprüfen
- technische Möglichkeiten überprüfen und demonstrieren
- frühzeitig mögliche Lösungsansätze präsentieren

Idee:
- rasche und billige Entwicklung eines prototypischen Systems als Diskussionsgrundlage

Typen unterscheiden sich in ...
- Lebensdauer: Wegwerfprototyp, evolutionärer Prototyp
- Zweck: technische Machbarkeit, Demonstration der Funktionalität oder Interaktion

Prototypen sind Zwischenprodukte in der Entwicklung, die dem Endprodukt in bestimmten relevanten Aspekten bereits stark ähneln sollen, die aber noch nicht vollständig sind. Eine andere Sichtweise von Prototypen ist es, sie als ein kostengünstigeres Modell für das Endprodukt zu betrachten.

Prototypen sind in anderen Ingenieursdisziplinen allgegenwärtig. Architekten lassen dreidimensionale maßstabsgetreue kleine Modelle eines Gebäudes anfertigen. Ein zukünftiger Bewohner kann auf diese Weise einen wesentlich lebendigeren Eindruck erhalten als es ihm mit einem Bauplan möglich wäre. Flugzeugbauer erproben an Prototypen den Windwiderstand im Windkanal.

Diese Idee übernimmt die Softwaretechnik. Die Ziele hierbei sind meist, die Anforderungen anhand eines Beispiels zu erheben und überprüfen, technische Möglichkeiten zu überprüfen und demonstrieren oder frühzeitig mögliche Lösungsansätze zu präsentieren (letztere Prototypen werden auch als *Demonstratoren* bezeichnet). Die Idee der Prototypen in der Softwaretechnik ist dieselbe wie bei anderen Ingenieursdisziplinen: die rasche und billige Entwicklung eines prototypischen Systems zur Erprobung.

Unter dem Begriff *Prototypen* versammeln sich in der Softwaretechnik aber sehr unterschiedliche Formen. Man kann sie unterscheiden anhand zweier unterschiedlicher Merkmale: zum einen anhand der Lebensdauer (Wegwerfprototyp, evolutionärer Prototyp), zum anderen anhand des Zwecks (Überprüfung der technischen Machbarkeit oder Demonstration der Funktionalität oder Interaktion).
Wir werden sie im Folgenden näher kennen lernen.

Typen von Prototypen in Bezug auf Lebensdauer

Wegwerf-Prototyp:

- beschreibt ein Softwaresystem exemplarisch
- dient zur Erhebung und Analyse von Anforderungen
- demonstriert die Funktionalität, die mit Stakeholdern diskutiert werden soll
- implementiert nicht notwendigerweise die gezeigte Funktionalität (z.B. GUI-Prototyp)
- ist als Komponente für das Endprodukt ungeeignet, weil billig erstellt

Typen von Prototypen in Bezug auf Lebensdauer

Wegwerf-Prototyp:

- beschreibt ein Softwaresystem exemplarisch
- dient zur Erhebung und Analyse von Anforderungen
- demonstriert die Funktionalität, die mit Stakeholdern diskutiert werden soll
- implementiert nicht notwendigerweise die gezeigte Funktionalität (z.B. GUI-Prototyp)
- ist als Komponente für das Endprodukt ungeeignet, weil billig erstellt

Der Wegwerfprototyp hat eine sehr begrenzte Lebensdauer. Er wird in keinem Fall zum Endprodukt werden. Sein Ziel ist es, einen Aspekt – sei es Funktionalität oder Interaktion – exemplarisch zu untersuchen oder zu demonstrieren. Dazu wird der relevante Aspekt mit möglichst billigen Mitteln realisiert (nicht notwendigerweise immer dadurch, dass tatsächlich Code geschrieben wird; siehe unten). Alle anderen Aspekte sind nicht realisiert. Weil der Prototyp unvollständig ist und mit billigen Mitteln erstellt wird, ist er für das Endprodukt ungeeignet – daher sein Name. Hat er den Aspekt demonstriert, wird er weggeworfen.

Typen von Prototypen in Bezug auf Lebensdauer

Evolutionärer Prototyp:

- dient zur schnellen Bereitstellung eines funktionsfähigen Systems im Rahmen von evolutionären Prozessmodellen zur Softwareentwicklung
- wird in weiteren Ausbaustufen zum endgültigen Produkt weiterentwickelt

Typen von Prototypen in Bezug auf Lebensdauer

Evolutionärer Prototyp:

- dient zur schnellen Bereitstellung eines funktionsfähigen Systems im Rahmen von evolutionären Prozessmodellen zur Softwareentwicklung
- wird in weiteren Ausbaustufen zum endgültigen Produkt weiterentwickelt

Im Gegensatz zum Wegwerf-Prototypen ist es beim evolutionären Prototypen vorgesehen, ihn schrittweise zum Endprodukt auszubauen. Er dient dazu, ein funktionsfähiges Systems möglichst schnell bereit zu stellen, wobei aber zu Anfang nicht die vollständige Funktionalität erwartet wird. Evolutionäre Prototypen finden ihren Einsatz unter anderem im Rahmen von evolutionären Prozessmodellen zur Softwareentwicklung.

Der Vorteil dieser inkrementellen Entwicklung ist es, dass jede funktionstüchtige Ausbaustufe vom Benutzer erprobt werden kann. Die Erfahrung und Kritik der Benutzer mit der Ausbaustufe nimmt Einfluss auf die nächste Ausbaustufe. Iterativ wird der evolutionäre Prototyp in weiteren Ausbaustufen zum endgültigen Produkt weiterentwickelt.

Bsp.: *Die Ausbaustufen für die evolutionäre Entwicklung eines Textverarbeitungssystems könnten zum Beispiel so aussehen:*

1. *grundlegende Funktionalität: Datei-Management, Editor, Textausgabe*
2. *erweiterte Funktionalität: Style-Files, Bearbeitung mathematischer Formeln, Einbinden von Graphiken*
3. *zusätzliche Funktionalität: Rechtschreibprüfung, Grammatiküberprüfung, Überarbeitungsmodus*
4. *ergänzende Funktionalität: Tabellenkalkulation, Geschäftsgraphiken, E-Mail, Web-Browser, Scanner-Anbindung*

□

Typen von Prototypen in Bezug auf Zweck

Technischer Prototyp:

- zeigt die technische Umsetzbarkeit von Ansätzen zur Problemlösung
- implementiert einen (kleinen) Ausschnitt der Funktionalität des Systems
- wird eher zur Machbarkeitsabschätzung und -demonstration eingesetzt

Typen von Prototypen in Bezug auf Zweck

Technischer Prototyp:
- zeigt die technische Umsetzbarkeit von Ansätzen zur Problemlösung
- implementiert einen (kleinen) Ausschnitt der Funktionalität des Systems
- wird eher zur Machbarkeitsabschätzung und -demonstration eingesetzt

Wegwerfprototypen und evolutionäre Prototypen unterscheiden sich anhand ihrer Lebensdauer. Ein völlig anderes Unterscheidungsmerkmal ist der Zweck, den man mit dem Prototypen verfolgt. Technische Prototypen zielen auf die technische Umsetzbarkeit von Ansätzen zur Problemlösung. Dazu implementiert der Prototyp einen (kleinen) Ausschnitt der Funktionalität des Systems. Technische Prototypen werden somit eher zur Machbarkeitsabschätzung und -demonstration eingesetzt. Sie gleichen dem Crash-Test-Stand, mit dessen Hilfe man das Verhalten von Karosserien im Automobilbau untersucht.

Technische Prototypen können sowohl für den Wegwurf oder den evolutionären Ausbau vorgesehen sein. Sie können sich des Weiteren unterschieden in Bezug auf die Softwarearchitektur.

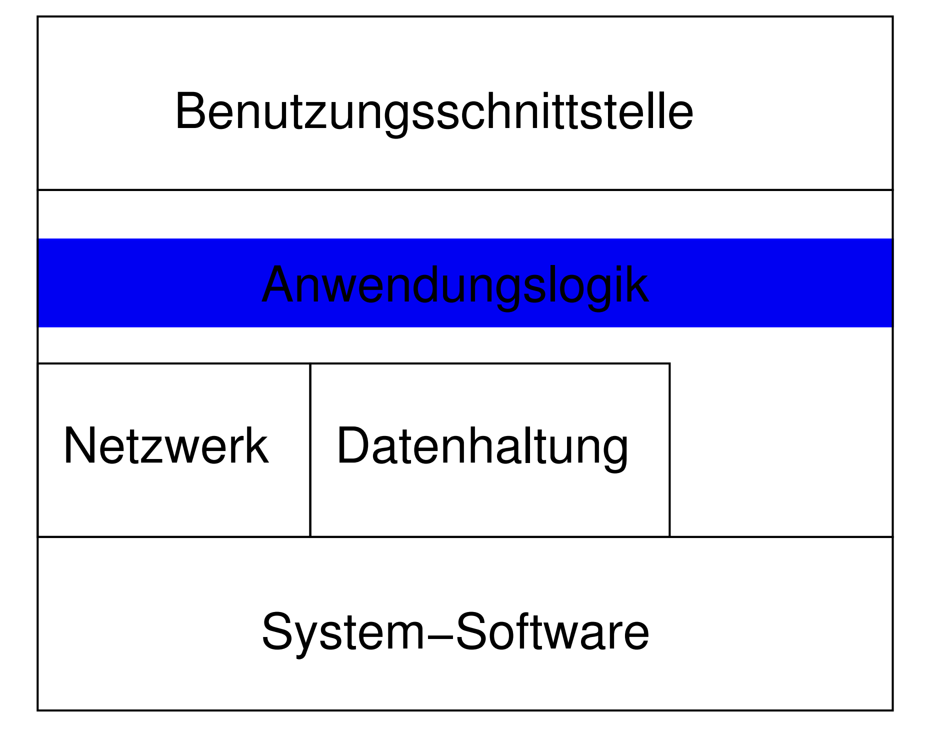
Horizontaler vs. vertikaler technischer Prototyp

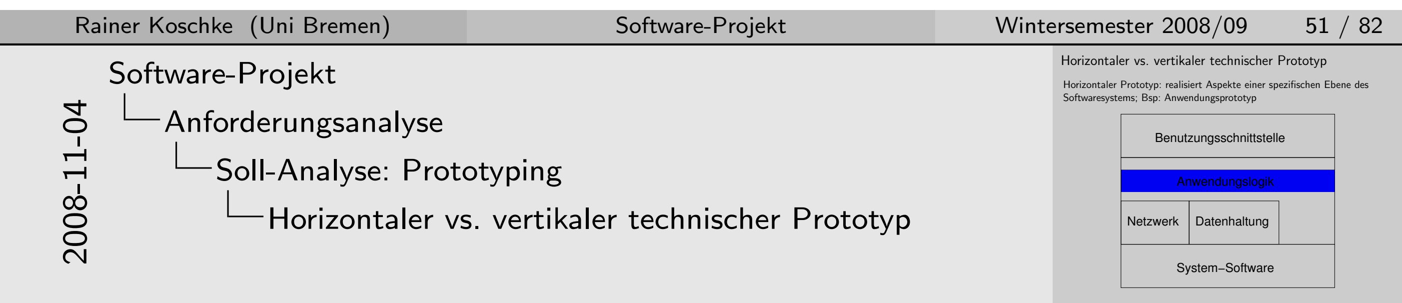
Horizontaler Prototyp: realisiert Aspekte einer spezifischen Ebene des Softwaresystems; Bsp: Oberflächenprototyp

Benutzungsschnittstelle
Anwendungslogik
Netzwerk
Datenhaltung
System–Software

Horizontaler vs. vertikaler technischer Prototyp

Horizontaler Prototyp: realisiert Aspekte einer spezifischen Ebene des Softwaresystems; Bsp: Anwendungsprototyp





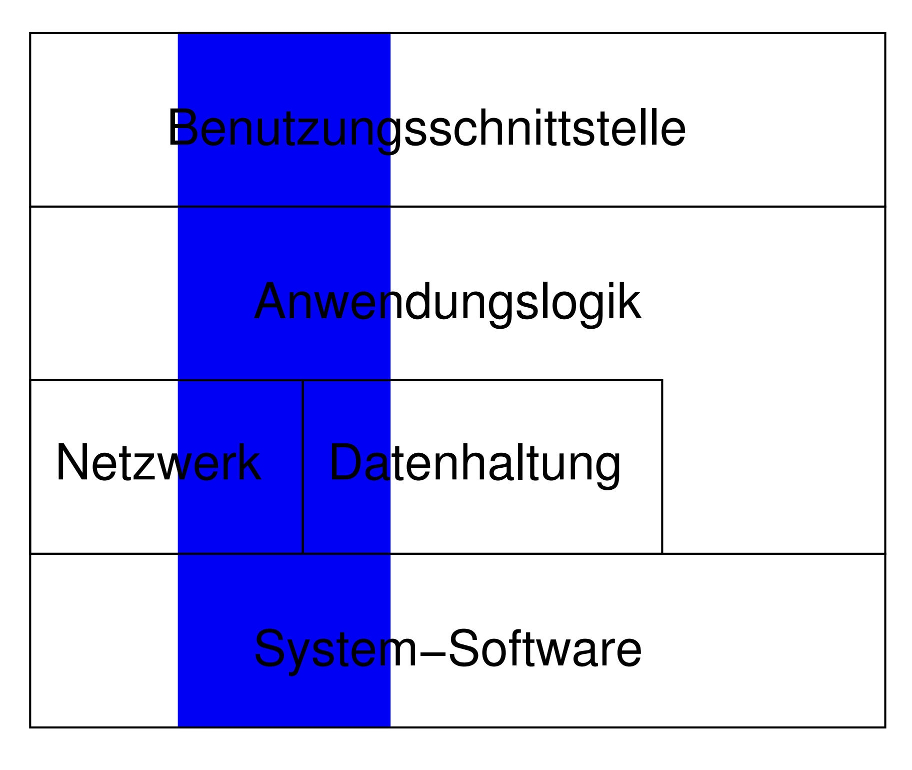
Softwaresysteme können zumindest konzeptionell in verschiedene Schichten eingeteilt werden, die jeweils von bestimmten Aspekten abstrahieren. Grob kann die Implementierung einteilen in die Schicht der Systemsoftware, die durch das Betriebssystem zur Verfügung gestellt wird, Schichten mit technischen Diensten, wie Netzwerkkommunikation oder Datenhaltung, sowie die Schicht, die die eigentlich anwendungsspezifische Logik implementiert und die Schicht der Benutzungsschnittstelle, die die Interaktion mit dem Benutzer übernimmt.

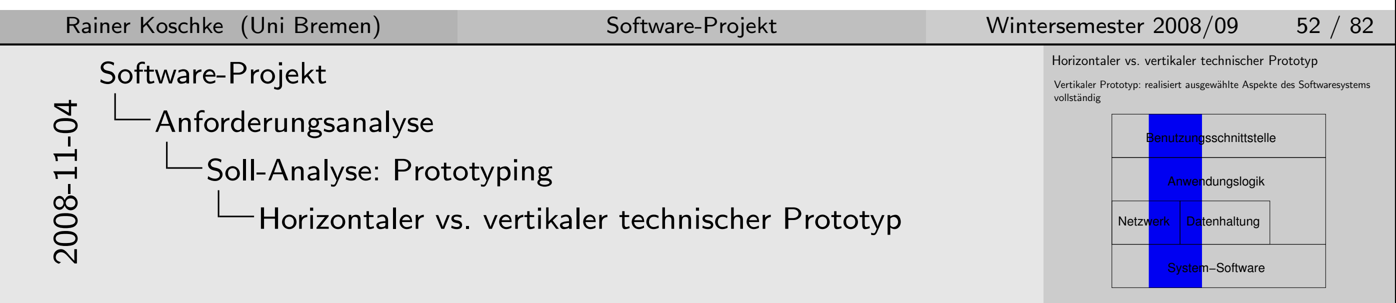
Hinsichtlich dieses konzeptionellen Aufbaus, kann man technische Prototypen weiter in horizontale und vertikale Prototypen unterscheiden.

Der **horizontale** technische Prototyp realisiert Aspekte einer spezifischen Ebene des Softwaresystems. Der Anwendungsprototyp implementiert zum Beispiel die Anwendungslogik und der Oberflächenprototyp die Benutzungsschnittstelle. Hiermit untersucht oder demonstriert man die Funktionalität einer ausgesuchten Schicht.

Da eine Schicht auf die andere aufbaut, aber nur eine Schicht implementiert wird, benötigen wir neben der Implementierung der relevanten Schicht selbst noch einen Simulator für die Schicht, auf die sich die relevante Schicht stützt. Man kann diese als Stumpf implementieren. Ein Stumpf bietet nicht die volle Funktionalität, sondern liefert auf die Anfragen der höheren Schicht einfach nur vorgefertigte Ergebnisse. Oberflächenprototypen können zum Beispiel so implementiert werden, dass die Anwendungsschicht alle Daten, die in der Oberfläche für bestimmte Anwendungsszenarien sichtbar sein sollen, in internen Tabellen abgespeichert hat und sie ohne weitere Berechnungen einfach als Ergebnis einer aufgerufenen Operation zurück liefert. Damit zeigt der Oberflächentyp möglicherweise falsche Daten an, aber zumindest die Interaktion lässt sich demonstrieren.

Vertikaler Prototyp: realisiert ausgewählte Aspekte des Softwaresystems vollständig





Der **vertikale** technische Prototyp realisiert ausgewählte Aspekte des Softwaresystems vollständig durch alle Schichten hindurch. Er stellt einen Durchstich durch das System dar. Mit seiner Hilfe lässt sich die Zusammenarbeit verschiedener Schichten untersuchen. Er findet auch bei inkrementeller Entwicklung Einsatz, bei der die erste Ausbaustufe funktionstüchtig ist (wenn auch nicht alle Funktionen implementiert sind). Dann wird der Durchstich in die Breite in den weiteren Ausbausstufen ausgedehnt, um die restliche Funktionalität zu implementieren. Man geht also zuerst in die Tiefe und dann in die Breite bei diesem Vorgehen.

Wir setzen solche Prototypen während der Anforderungsanalyse ein, wenn wir kritische Anforderungen haben, von denen wir nicht sicher sind, dass wir sie realisieren können. Ansonsten werden sie auch gerne vor Entwurf der Architektur angewandt, um einzusetzende Technologien oder geplante Entwürfe zu untersuchen.

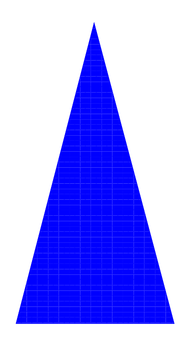
Prototypen für Interaktion: Storyboards

Storyboards

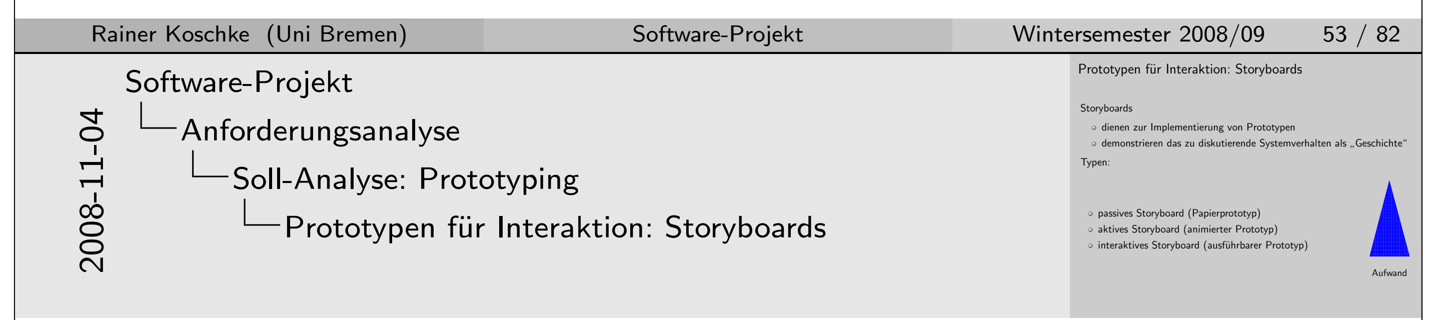
- dienen zur Implementierung von Prototypen
- demonstrieren das zu diskutierende Systemverhalten als „Geschichte"

Typen:

- passives Storyboard (Papierprototyp)
- aktives Storyboard (animierter Prototyp)
- interaktives Storyboard (ausführbarer Prototyp)



Aufwand



In der Anforderungsanalyse müssen wir die Interaktion zwischen Benutzer und Anwender festlegen. Dieser Interaktion nimmt eine Schlüsselrolle ein, weil ein Softwaresystem, das alle gewünschten Funktionalitäten bietet, dennoch unbrauchbar sein kann, wenn man es nicht benutzen kann. Aus diesem Grunde wollen schon sehr früh, Interaktion ausprobieren können und sie Kunden und Benutzern vorstellen. Hier können wir Oberflächenprototypen entwickeln.

Oberflächenprototypen sind horizontale Prototypen, bei denen die Interaktion untersucht oder demonstriert werden soll. Meist dienen sie weniger der technischen Machbarkeit, sondern vielmehr der konzeptionellen Angemessenheit und Schlüssigkeit. Wir untersuchen damit, ob der Benutzer später mit unserem System effizient und effektiv interagieren kann.

Dem Oberflächenprototypen kommt unter den Prototypen eine Sonderstellung zu, weil es die Oberfläche ist, die der Benutzer später sehen und bedienen wird. Alle anderen Schichten sind für ihn unsichtbar. Wenn wir die von uns einzusetzenden Implementierungsechnologien bereits sicher beherrschen, können wir auf solche technische Prototypen tieferer Schichten verzichten. Weil wir aber in der Anforderungsspezifikation auch die Benutzungsschnittstelle beschreiben müssen und die Qualität einer Software aus Benutzersicht steht und fällt mit ihrer Benutzbarkeit, sind Oberflächenprototypen für die Anforderungsanalyse unerlässlich.

Aus diesem Grunde genießen Oberflächenprototypen hier unser Hauptaugenmerk. Wir unterscheiden dabei die folgenden Typen:

- passives Storyboard (Papierprototyp)
- aktives Storyboard (animierter Prototyp)
- interaktives Storyboard (ausführbarer Prototyp)

Alle Typen von Storyboards beschreiben die zukünftige Interaktion mit dem System als eine Art Drehbuch, daher ihr Name. Sie unterscheiden sich jedoch im Grad ihrer Interaktivität. Das interaktive Storyboard bietet dabei die höchste Interaktivität. Mit dem Grad der Interaktivität steigen aber auch die Kosten für die Entwicklung des Storyboards. Wir werden im Folgenden diese drei Typen näher beschreiben.

Passive Storyboards

Demonstration:

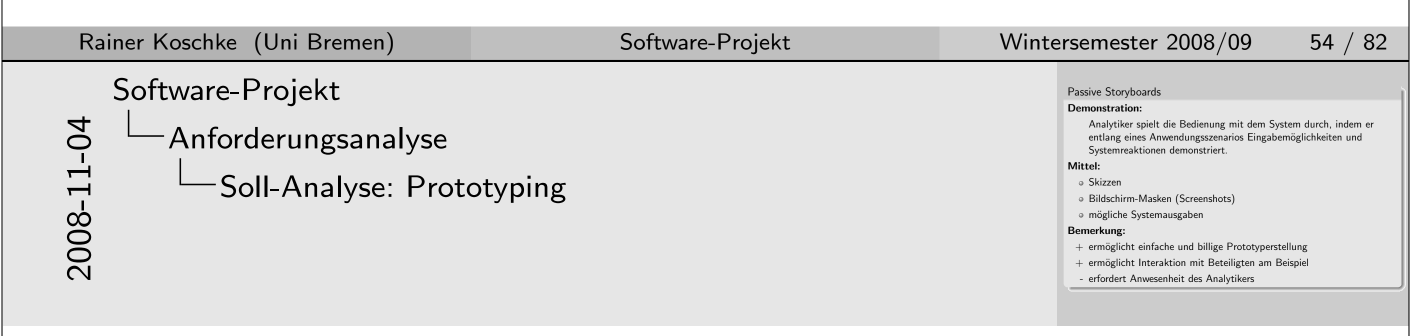
Analytiker spielt die Bedienung mit dem System durch, indem er entlang eines Anwendungsszenarios Eingabemöglichkeiten und Systemreaktionen demonstriert.

Mittel:

- Skizzen
- Bildschirm-Masken (Screenshots)
- mögliche Systemausgaben

Bemerkung:

+ ermöglicht einfache und billige Prototyperstellung
+ ermöglicht Interaktion mit Beteiligten am Beispiel
- erfordert Anwesenheit des Analytikers



Passive Storyboards erlauben selbst keine Interaktion. Statt dessen spielt der Analytiker bei der Demonstration die Bedienung mit dem System durch, indem er entlang eines Anwendungsszenarios die Eingabemöglichkeiten und Systemreaktionen demonstriert.

Als Mittel werden Skizzen, Bildschirm-Masken in Form von Bildschirmabzügen (Screenshots) und mögliche Systemausgaben verwendet. Die einfachsten Materialien sind Papier und Bleistift. Dafür sprechen die besonders einfache und billige Prototyperstellung sowie die Möglichkeit zur Interaktion mit den Beteiligten am konkreten Beispiel. Der Benutzer selbst kann mit diesen Materialien selbst umgehen und an der Entwicklung des Prototypen direkt mitwirken.

Passive Storyboards

Demonstration:

Analytiker spielt die Bedienung mit dem System durch, indem er entlang eines Anwendungsszenarios Eingabemöglichkeiten und Systemreaktionen demonstriert.

Mittel:

- Skizzen
- Bildschirm-Masken (Screenshots)
- mögliche Systemausgaben

Bemerkung:

+ ermöglicht einfache und billige Prototyperstellung
+ ermöglicht Interaktion mit Beteiligten am Beispiel
- erfordert Anwesenheit des Analytikers

Bsp.: *Unsere Analyse der Schwachstellen in der Ist-Analyse zu unserem Fahrradladenbeispiel hat ergeben, dass es oft schwierig ist, einen Kunden beim Einkauf eines Artikels für ein Fahrrad, das sich schon in seinem Besitz befindet, richtig zu beraten, weil er nicht weiß, welche Daten das Fahrrad hat. Wenn er einen Flaschenhalter kaufen möchte, so muss er zu den Ausmaßen und Bohrungen sowie der Farbe des Rahmens passen. Wenn er sie nicht kennt, kann ihm der Verkäufer keine Empfehlung aussprechen.*

Wir haben uns entschlossen, einen Einkaufsassistenten auf Basis eines PDAs zu entwickeln. Auf diesem tragbaren Kleinrechner sind alle Einkäufe des Kunden erfasst und zwar branchenübergreifend, also Fahrradteile genauso wie Computer-Hardware etc. Wir möchten dem Benutzer die zukünftige Interaktion mit einem solchen PDA demonstrieren. Die Interaktion mit einem PDA hat besondere Herausforderungen, weil die darstellbare Fläche sehr begrenzt und typische Eingabegeräte wie Maus und Tastatur nicht vorhanden sind.

Bild 1: *Wir fertigen einen Papierprototypen an, indem wir einen realen PDA hernehmen und die dargestellten Elemente zeichnen und auf das Display legen. Das Bild zeigt die Einstiegsmaske mit den Produktgruppen des Käufers, die mit dem PDA verwaltet werden. Im Fahrradladen wählt er das Fahrradsymbol aus, wenn er einen Artikel für sein Fahrrad kaufen möchte. Dadurch gelangt er zum nächsten Dialog.* □

Beispiel passives Storyboard



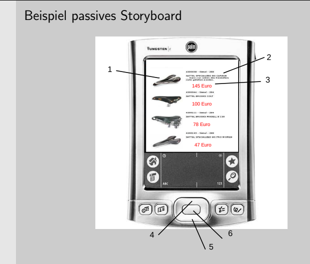


Bsp.: **Bild 2:** *Im nächsten Dialog wird sein Fahrrad angezeigt. Er kann dann mit dem Stift auf ein Bestandteil seines Rades tippen. Darauf hin wird ein Menü eingeblendet, in dem er die nächste Aktion auswählen kann. Er kann einen Artikel abfragen, um Daten über ihn abzufragen, ändern, um den Artikel in seinen Eigenschaften zu ändern, entfernen, um ihn aus seiner Konfiguration zu löschen, oder ersetzen, um ihn durch einen neuen Artikel zu ersetzen.*
□

Beispiel passives Storyboard



Beispiel passives Storyboard



Bsp.: **Bild 3:** *Nehmen wir an, wir wählen den Sattel aus, und wählen* $ersetzen$. *Dann gelangen wir zum nächsten Dialog. Als nächstes werden alle Sättel angezeigt, die zum Fahrrad des Kunden passen, sortiert nach dem Preis. Es werden maximal vier Artikel angezeigt. Diese Liste kann weiter verarbeitet werden, indem wir mit den Stift tippen oder die Tasten bedienen. Berühren wir mit dem Stift das Bild des Sattels (1), den Beschreibungstext (2) oder den Preis, dann gelangen wir zum Kaufdialog. Bedienen wir auf dem Tastenfeld die Nach-Oben- (4) beziehungsweise die Nach-Unten-Taste (5) können wir den sichtbaren Ausschnitt in der Liste der gefundenen Artikel nach oben beziehungsweise nach unten bewegen. Mit der Auswahltaste (5) gelangen wir zum Kaufdialog.* □

Beispiel passives Storyboard

Screen Auswahl

Screen Kaufen

145 Euro

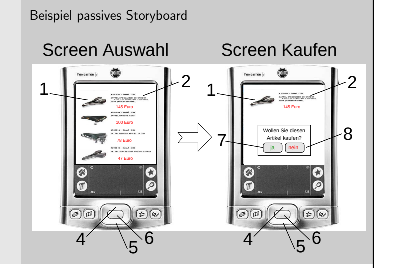
100 Euro

78 Euro

47 Euro

Wollen Sie diesen Artikel kaufen?

ja nein



Bsp.: **Bild 4:** *Im Kaufdialog wird nur der ausgewählte Artikel angezeigt und der Kunde wird gefragt, ob er den Artikel kaufen möchte. Bestätigt er mit der Schaltfläche "Ja" (7), dann wird der Artikel in den virtuellen Warenkorb des elektronischen Verkaufsassistenten gelegt, der vom Verkäufer an der Kasse ausgelesen werden kann. Mit der Schaltfläche "Nein" wird der Vorgang abgebrochen, und man gelangt zum Auswahldialog zurück.*

Die Entscheidung, maximal vier Artikel anzuzeigen, haben wir in der Vorbereitung des Papierprototypen getroffen. Wir hatten sehr schnell im maßstabgetreuen Modell feststellen können, dass mehr als vier Artikel nicht auf dem kleinen Monitor zu präsentieren sind. Wir haben jedoch versäumt, irgendwie kenntlich zu machen, dass mehr als vier Artikel als Ergebnis der Suchanfrage geliefert wurden, aber nicht alle sichtbar sind. Der Kunde fragt uns während der Vorführung, wie das denn zu erkennen sei. Der Kunde nimmt dann einen roten Farbstift, zeichnet dafür rote Pfeile nach oben beziehungsweise unten über dem ersten beziehungsweise letzten angezeigten Element ein und sagt uns, dass er das gerne so hätte. Während wir dem Kunden diesen Prototypen vorführen, bemerkt dieser außerdem, dass wir im Auswahldialog keine Möglichkeit vorgesehen haben, den Dialog durch einen anderen Weg als zum Kaufdialog zu verlassen. Uns ist also glücklicherweise früh genug vom Kunden klar gemacht worden, dass wir in der Dialogführung einige Dinge übersehen haben. Die Überarbeitung unseres Papierprototypen ist aber einfach. So einfach, dass es der Kunde gleich selbst übernehmen konnte. □

Passive Storyboards eignen sich besonders in der frühen Phase des Interaktions-Designs, in der man den Benutzer aktiv in den Design-Prozess einbeziehen möchte.

Aktive Storyboards

Demonstration:

Abspielen einer selbstablaufenden Präsentation des Systemverhaltens.

Mittel:

- Film, Diashow
- selbstablaufende PowerPoint-Präsentation

Bemerkung:

+ ermöglicht einfache automatisierte Darstellung von typischen Anwendungsszenarien
+ erfordert nicht unbedingt die Anwesenheit von Analytikern
- erlaubt keine Interaktion während der Präsentation

Aktive Storyboards

Demonstration:

Abspielen einer selbstablaufenden Präsentation des Systemverhaltens.

Mittel:

- Film, Diashow
- selbstablaufende PowerPoint-Präsentation

Bemerkung:

+ ermöglicht einfache automatisierte Darstellung von typischen Anwendungsszenarien
+ erfordert nicht unbedingt die Anwesenheit von Analytikern
- erlaubt keine Interaktion während der Präsentation

Während passive Storyboards durch den Analytiker demonstriert werden müssen, sind aktive Storyboards Präsentationen des Systemverhaltens, die von selbst ablaufen. Sie laufen wie ein Film ab. Dazu kann man tatsächlich kontinuierliche Sequenzen wie in einem Film darstellen (z.B. mit Werkzeugen wie Macromedia Flash oder Camtesia) oder aber Einzelbilder der Dialoge (z.B. mit einer Präsentationssoftware wie OpenOffice oder Powerpoint) anzeigen.

Typische Anwendungsszenarien können automatisiert dargestellt werden. Weil ein aktives Storyboard von allein abläuft, ist die Anwesenheit von Analytikern nicht unbedingt erforderlich. Damit können solche Demos an eine Vielzahl von möglichen Benutzern verschickt und deren Meinung eingeholt werden. Allerdings erlauben aktive Storyboards keine Interaktion während der Präsentation, was gerade in der frühen Phase des Interaktions-Designs wünschenswert ist.

Aktive Storyboards eignen sich besonders in einer fortgeschritteneren Phase des Interaktions-Designs, in der man ein ausgearbeitetes Konzept sehr vielen Benutzern vorstellen möchte.

Interaktive Storyboards

Demonstration:

Prototyp ermöglicht dem Benutzer die frühzeitige Interaktion mit dem möglichen System; Funktionalität kann evtl. durch Analytiker „von Hand“ simuliert werden

Mittel:

- ausführbares Programm, das Teile der Funktionalität realisiert

Bemerkung:

+ ermöglicht Interaktion des Nutzers mit dem System
+ erlaubt größtmögliche Nähe zum realen System
- erfordert höheren Aufwand bei Prototyp-Erstellung

Interaktive Storyboards
Demonstration:
Prototyp ermöglicht dem Benutzer die frühzeitige Interaktion mit dem möglichen System; Funktionalität kann evtl. durch Analytiker „von Hand" simuliert werden
Mittel:
- ausführbares Programm, das Teile der Funktionalität realisiert

Bemerkung:
+ ermöglicht Interaktion des Nutzers mit dem System
+ erlaubt größtmögliche Nähe zum realen System
- erfordert höheren Aufwand bei Prototyp-Erstellung

Interaktive Storyboards bieten die größtmögliche Interaktion mit dem System. Dazu wird ein ausführbares Programm entwickelt, was die Eigenschaften der Benutzungsschnittstelle demonstriert. Es können einzelne Teile der Funktionalität bereits prototypisch implementiert sein. Meist genügen aber auch vorbereitete Daten für bestimmte Szenarien, die nicht wirklich berechnet, sondern nur in Reaktion auf die Eingabe des Benutzers angezeigt werden. Die Daten müssen auch nicht korrekt sein, solange es nur um die Demonstration der Interaktion geht. Die Funktionalität kann unter Umständen auch durch den Analytiker „von Hand" simuliert werden.

Interaktive Storyboards ermöglichen echte Interaktion des Nutzers mit dem System mit einer größtmöglichen Nähe zum realen System. Dabei muss der Analytiker nicht bei der Demonstration anwesend sein. Allerdings kann der Benutzer auch hier nicht selbst eingreifen, um eigene Vorstellungen einzubringen, und wir haben einen höheren Aufwand bei der Prototyp-Erstellung. GUI-Design-Werkzeuge können jedoch einen Teil des Aufwands reduzieren.

Interaktive Storyboards werden in der Regel in späteren Phasen des Interaktions-Designs entwickelt, bei denen man schon genaue Vorstellungen entwickelt hat. Sie können für Nutzertests zur Gebrauchstauglichkeit des Systems eingesetzt werden, indem man verschiedene Nutzer (pro Persona mindestens einen) mit dem System interagieren lässt und diese Interaktion beobachtet und auswertet.

Vor- und Nachteile des Einsatzes von Prototypen

+ erlauben frühzeitige Demonstration von Lösungsansätzen
+ erlauben frühzeitige Beteiligung der Benutzer
+ vermeiden das „Leere-Blatt-Syndrom"
+ reduzieren Entwicklungsrisiken durch frühzeitige Diskussion mit Beteiligten
+ geeignete Werkzeuge ermöglichen die schnelle Erstellung von Prototypen
– erfordern erhöhten Entwicklungsaufwand durch (zusätzliche) Prototyp-Entwicklung
– Gefahr, dass Wegwerf-Prototyp Teil des Produkts wird (z.B. aus Zeitdruck)

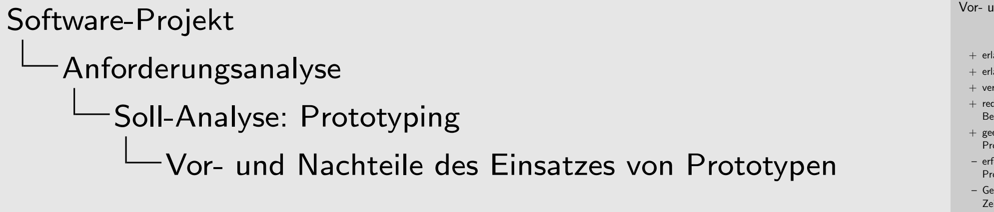
Vor- und Nachteile des Einsatzes von Prototypen

+ erlauben frühzeitige Demonstration von Lösungsansätzen
+ erlauben frühzeitige Beteiligung der Benutzer
+ vermeiden das „Leere-Blatt-Syndrom"
+ reduzieren Entwicklungsrisiken durch frühzeitige Diskussion mit Beteiligten
+ geeignete Werkzeuge ermöglichen die schnelle Erstellung von Prototypen
− erfordern erhöhten Entwicklungsaufwand durch (zusätzliche) Prototyp-Entwicklung
− Gefahr, dass Wegwerf-Prototyp Teil des Produkts wird (z.B. aus Zeitdruck)

Zum Ende des Abschnitts über Prototypen fassen wir die Vor- und Nachteile von Prototypen zusammen.
Vorteile:

+ Prototypen erlauben eine frühzeitige Demonstration von Lösungsansätzen mit vertretbaren Kosten. Gerade weil der Interaktion eine hohe Bedeutung zukommt und sie auch in der Anforderungsspezifikation genau beschrieben sein sollte, sollten wir in der Anforderungsanalyse präzise Konzepte entwickeln und sie dem Kunden und Benutzer vorführen, um d Konzepte zu evaluieren und zu verbessern. Andererseits können wir zu einem so frühen Zeitpunkt nicht die ganze Applikation bereits entwickeln. Darum also Prototypen.
+ Prototypen erlauben damit eine frühzeitige Beteiligung der Benutzer. Der Benutzer selbst kann sich ein genaueres Bild des späteren Systems machen und frühzeitig korrigierend eingreifen.
+ Prototypen vermeiden das „Leere-Blatt-Syndrom", das wir von Schriftstellern kennen, die die erste Seite ihres neuen Romans schreiben sollen. Mit Prototypen können wir direkt loslegen. Insbesondere Papierprototypen helfen uns einen schnellen Einstieg zu finden, weil sie billig herzustellen und leicht änderbar sind.
+ Prototypen reduzieren Entwicklungsrisiken durch frühzeitige Diskussion mit Beteiligten. Wir wissen, wie teuer uns spät Änderungen kommen. Darum stellen wir durch frühes Vorzeigen und Einbeziehen der Benutzer sicher, dass wir auf dem richtigen Weg sind. Dafür ist eine Anfangsinvestition notwendig, um den Prototypen zu entwickeln, die sich aber im Verlaufe des Projekts mit sehr hoher Wahrscheinlichkeit amortisiert, weil weniger Fehler in der Anforderungsanalyse gemacht werden.
+ Geeignete Werkzeuge ermöglichen die schnelle Erstellung von Prototypen, so dass sich die Kosten in Grenzen halten. Beispielsweise gibt es GUI-Builder, mit denen man sich rasch graphische Benutzeroberflächen zusammen klicken kann. Oder man benutzt Werkzeuge wie Graphikprogramme (auch scripting-fähige, wie Adobe Flash) und Präsentationssoftware.

2008-11-04



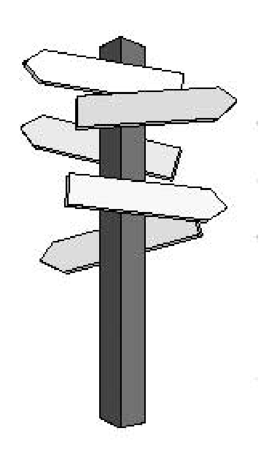
Vor- und Nachteile des Einsatzes von Prototypen

+ erlauben frühzeitige Demonstration von Lösungsansätzen
+ erlauben frühzeitige Beteiligung der Benutzer
+ vermeiden das „Leere-Blatt-Syndrom"
+ reduzieren Entwicklungsrisiken durch frühzeitige Diskussion mit Beteiligten
+ geeignete Werkzeuge ermöglichen die schnelle Erstellung von Prototypen
− erfordern erhöhten Entwicklungsaufwand durch (zusätzliche) Prototyp-Entwicklung
− Gefahr, dass Wegwerf-Prototyp Teil des Produkts wird (z.B. aus Zeitdruck)

Nachteile:

– Auch wenn Prototypen billiger sind als das Endprodukt, so erfordern Prototypen einen erhöhten Entwicklungsaufwand durch die zusätzlich Prototyp-Entwicklung. Je nach Ausbau des Prototypen können die Kosten erheblich sein.
– Bei der Entwicklung von Wegwerfprototypen wird kein großer Wert auf Qualität gelegt; sie müssen zeigen, was sie zeigen sollen, und ansonsten möglichst billig sein. Der größte Nachteil dieser Prototypen ist somit die Gefahr, dass sie zum Teil des Produkts werden (z.B. aus Zeitdruck). Bei interaktiven Storyboards erhalten Kunde und Projektmanager den Eindruck, dass das System schon beinahe fertig ist. Die Versuchung ist groß, den Wegwerfprototypen auszubauen statt ihn wie geplant wegzuwerfen. Das rächt sich jedoch in der Regel, weil die Qualität des Wegwerfprototypen ein solches Vorgehen nicht unterstützt. Dieser Gefahr kann man durch Papierprototypen oder durch eine Entwicklung mit einer Programmiersprachen, die man später für das Endprodukt nicht benutzen möchte, vorbeugen.

Wegweiser



Analysetechniken

Wann wird welche Analysetechnik eingesetzt?

Analysetechniken

	Ist-Zustand	**Soll-Zustand**	**Folgen**
Auswertung vorhandener Daten/Dokumente	+	-	-
Beobachtungen	+	○	-
Befragung			
- geschlossene Fragen	+	○	-
- offene Fragen	+	○	-
- hybride Fragen	+	○	-
Prototyping	-	+	+
partizipative Entwicklung	-	+	+

Analysetechniken

	Ist-Zustand	Soll-Zustand	Folgen
Auswertung vorhandener Daten/Dokumente	+	-	-
Beobachtungen	+	○	-
Befragung			
- geschlossene Fragen	+	○	-
- offene Fragen	+	○	-
- hybride Fragen	+	○	-
Prototyping	-	+	+
partizipative Entwicklung	-	+	+

Zum Abschluss der Ist- und Sollanalyse fassen wir noch einmal die Techniken, die wir kennen gelernt haben, zusammen und bewerten ihre Eignung für das Erfassen des Ist-Zustands, der Planung des Soll-Zustands und der Abschätzung mittel- und langfristiger Folgen unserer anvisierten Softwarelösung. Die folgende Tabelle gibt einen Überblick:

	Ist-Zustand	Soll-Zustand	Folgen
Auswertung vorhandener Daten/Dokumente	+	-	-
Beobachtungen	+	○	-
Befragung			
- geschlossene Fragen	+	○	-
- offene Fragen	+	○	-
- hybride Fragen	+	○	-
Prototyping	-	+	+
partizipative Entwicklung	-	+	+

Die Auswertung vorhandener Dokumente ist ein Instrument rein für die Erhebung des Ist-Zustands, weil sie auf Existierendem basiert. Befragung und Beobachtung setzen auch etwas Existierendes voraus. Man kann sie aber auch einsetzen, um Prototypen zu evaluieren. Prototypen sind eine Vision des Soll-Zustandes, die handfest und überprüfbar ist. Mit ihnen kann man durch geeignete Evaluierung auch die Folgen des Einsatzes des späterens Systems abschätzen. Man kann also zum Beispiel demonstrieren, dass frühere Redundanzen und manuelle Eingriffe nicht mehr existieren werden, was die Fehleranfälligkeit reduziert. Außerdem kann man nachweisen, dass ein Benutzer schneller seine Aufgabe erledigen kann, als das vorher der Fall war.



2008-11-04

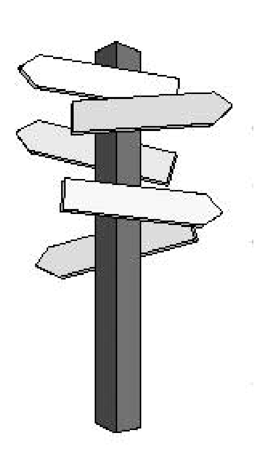
Software-Projekt
└─Anforderungsanalyse
└─Zusammenfassung der Techniken
└─Analysetechniken

Analysetechniken

	Ist-Zustand	Soll-Zustand	Folgen
Auswertung vorhandener Daten/Dokumente	+	-	-
Beobachtungen	+	○	-
Befragung			
- geschlossene Fragen	+	○	-
- offene Fragen	+	○	-
- hybride Fragen	+	○	-
Prototyping	-	+	+
partizipative Entwicklung	-	+	+

Die partizipative Entwicklung ist die Softwareentwicklung, die die späteren Benutzer einbezieht – wohl gemerkt, die Benutzer, nicht nur den Kunden. Es gibt hierfür verschiedene Ausprägungen. Die Benutzer können zu Anfang in der Phase des Interaktions-Designs bei der Entwicklung von Prototypen einbezogen werden und gegen Ende beim Akzeptanztest. Bei inkrementellen Entwicklungsprozessen, bei denen ein System in kleineren funktionstüchtigen Inkrementen entwickelt werden, werden sie noch stärker einbezogen. Dort evaluieren sie jedes einzelne Inkrement – also Zwischenprodukt – und nicht nur das Endprodukt. Da die Anforderungsanalyse bei der Entwicklungsphase des nächsten Inkrements wieder durchlaufen wird, kann der Benutzer an vielen Punkten der Entwicklung Einfluss nehmen. Diese Idee wird beim *Extreme Programming*, einem agilen Entwicklungsprozess, zum Extrem getrieben, indem ein Benutzer vor Ort der Entwicklung ist und wöchentlich das System testet und Verbesserungsvorschläge unterbreitet. Die Gebrauchstauglichkeit kann auf diese Weise besser sicher gestellt werden. Außerdem haben Benutzer die Möglichkeit, das System, das sie später einsetzen sollen, mitzugestalten, was zu einer höheren Akzeptanz führen sollte.

Wegweiser



Anforderungsspezifikation

Wie halten wir die Anforderungen fest?

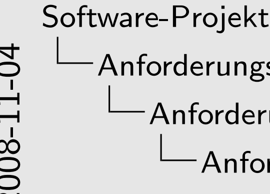
Anforderungsspezifikation nach IEEE Std 610.12-1990

Definition

requirement: condition or capability needed by a user to solve a problem or achieve an objective.

specification: document that specifies, in a complete, precise, verifiable manner, the requirements (, ...) of a system or component, and, often, the procedures for determining whether these provisions have been satisfied.

software requirements specification (SRS): documentation of the essential requirements (functions, performance, design constraints, and attributes) of the software and its external interfaces.



Anforderungsspezifikation nach IEEE Std 610.12-1990

Definition

requirement: condition or capability needed by a user to solve a problem or achieve an objective.

specification: document that specifies, in a complete, precise, verifiable manner, the requirements (, ...) of a system or component, and, often, the procedures for determining whether these provisions have been satisfied.

software requirements specification (SRS): documentation of the essential requirements (functions, performance, design constraints, and attributes) of the software and its external interfaces.

An die Ist- und Sollanalyse schließt sich die Formulierung der Anforderungen an. Die Sollanalyse ergibt, was die Software leisten muss, um die Schwachstellen, die durch die Ist-Analyse offenbar wurden, zu beseitigen. Die Sollanalyse hat jedoch nur die prinzipiellen Konzepte erarbeitet. Diese müssen wir nun präzise ausarbeiten und am besten schriftlich festhalten. In diesem Abschnitt setzen wir uns deshalb mit der Frage auseinander, wie die Anforderungen formuliert sein sollen. Bevor wir jedoch dazu kommen, müssen wir über ein paar grundlegende Begriffe einig werden. Was ist denn eine Anforderungsspezifikation genau?

Im Englischen wird die Anforderungsspezifikation *Requirement Specification* genannt, was sich aus *Requirement* für Anforderung und *Specification* zusammen setzt. Das Software-Engineering-Glossar IEEE Std 610.12-1990 definiert *Requirement* wie folgt:

Requirement : condition or capability needed by a user to solve a problem or achieve an objective.

Zu Deutsch: eine Anforderung ist eine Bedingung oder Fähigkeit, die von einem Benutzer benötigt wird, um ein Problem zu lösen oder ein Ziel zu erreichen.

Eine Spezifikation ist wie folgt definiert:

Specification : document that specifies, in a complete, precise, verifiable manner, the requirements (, ...) of a system or component, and, often, the procedures for determining whether these provisions have been satisfied.

Eine Spezifikation ist ein Dokument, das vollständig, präzise und überprüfbar die Anforderungen an ein System oder eine Komponente spezifiziert und oft auch die Prüfprozeduren nennt, um festzustellen, ob diese Forderungen tatsächlich erfüllt sind.

Diese Definition erläutert nicht nur den Begriff *Spezifikation*, sondern stellt auch eine Reihe von Forderungen für Spezifikation auf. Sie sollen vollständig, präzise und überprüfbar sein. Vollständig bedeutet, dass die Spezifikation alle relevanten Anforderungen nennt. Präzise heißt, dass keine Anforderung missverstanden werden kann. Überprüfbar bedeutet, dass man für jede Anforderung eine Prüfprozedur angeben kann, anhand derer man objektiv nachweisen kann, dass eine Anforderung erfüllt ist. Wir werden uns mit diesen Eigenschaften und mit weiteren Qualitäten, die eine gute Anforderungsspezifikation auszeichnen, in diesem Abschnitt noch auseinander setzen.

2008-11-04
Software-Projekt
└─Anforderungsanalyse
└─Anforderungsspezifikation
└─Anforderungsspezifikation nach IEEE Std 610.12-1990



Anforderungsspezifikation nach IEEE Std 610.12-1990

Definition

requirement: condition or capability needed by a user to solve a problem or achieve an objective.

specification: document that specifies, in a complete, precise, verifiable manner, the requirements (, ...) of a system or component, and, often, the procedures for determining whether these provisions have been satisfied.

software requirements specification (SRS): documentation of the essential requirements (functions, performance, design constraints, and attributes) of the software and its external interfaces.

Der zusammengesetzte Begriff *Software Requirements Specification* ist wie folgt definiert:

Software Requirements Specification : documentation of the essential requirements (functions, performance, design constrain and attributes) of the software and its external interfaces.

Diese Definition ergänzt die einzelnen Definitionen von *Anforderung* und *Spezifikation* um den Inhalt, der in einer Anforderungsspezifikation aufgeführt ist. Die Anforderungsspezifikation ist demnach eine Dokumentation der wesentlichen Anforderungen (Funktionen, Performanz, Entwurfseinschränkungen, und Attribute) einer Software sowie ihrer externen Schnittstellen. Diese Inhalte entstammen einer anderen IEEE-Norm, die beschreibt, wie Anforderungsspezifikation aufgebaut sind und welche Inhalte sie haben. Mit dieser Norm werden wir uns hier noch auseinander setzen.

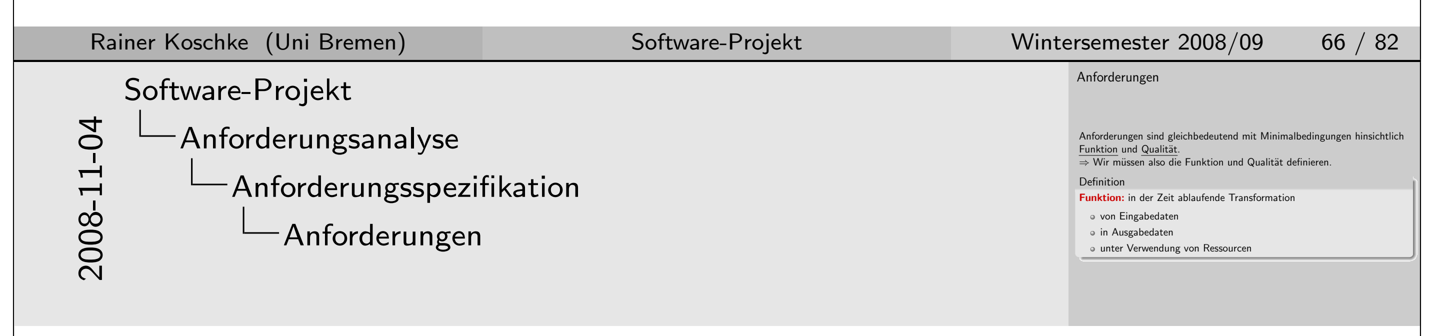
Anforderungen

Anforderungen sind gleichbedeutend mit Minimalbedingungen hinsichtlich Funktion und Qualität.
⇒ Wir müssen also die Funktion und Qualität definieren.

Definition

Funktion: in der Zeit ablaufende Transformation

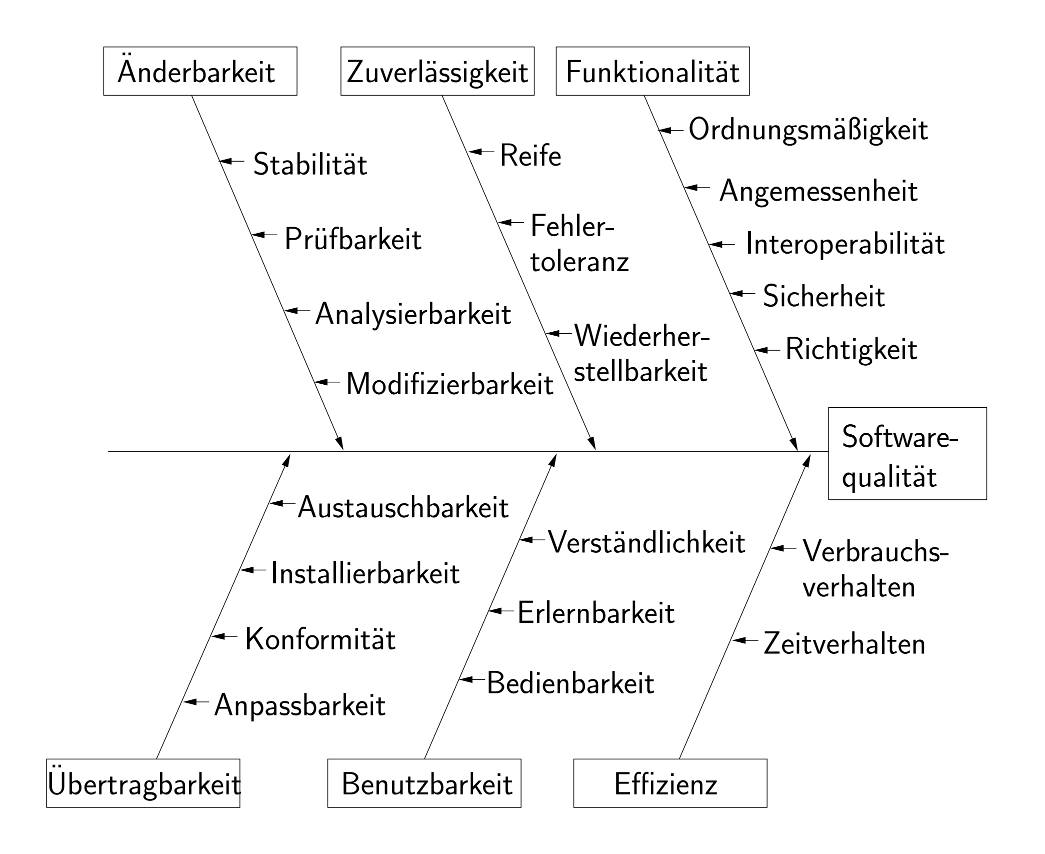
- von Eingabedaten
- in Ausgabedaten
- unter Verwendung von Ressourcen

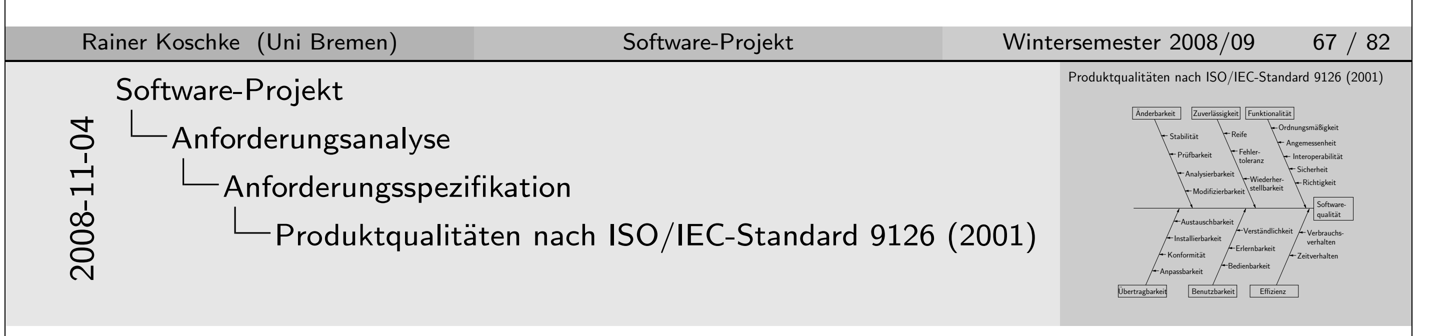


Anforderungen legen die Minimalbedingungen hinsichtlich Funktion und Qualität fest. Was aber sind *Funktionen* und wie definiert man *Qualität* für Software?

Während sich Funktionen über den Zusammenhang von Eingaben des Benutzers und darauf hin erwartete Ausgaben unter Verwendung einer maximalen Menge von Ressourcen definieren lassen, ist der Begriff *Softwarequalität* weniger offensichtlich.

Produktqualitäten nach ISO/IEC-Standard 9126 (2001)





Der Begriff *Softwarequalität* ist sehr vielschichtig. Im Allgemeinen bedeutet Qualität lediglich, zu welchem Grad ein Gegenstand bestimmte Attribute aufweist. Während die Qualität einer Schraube recht einfach durch die Attribute Festigkeit, Gewicht, Material, Preis etc. festgelegt werden kann, sind die wesentlichen Attribute bei Software nicht ohne Weiteres offensichtlich.

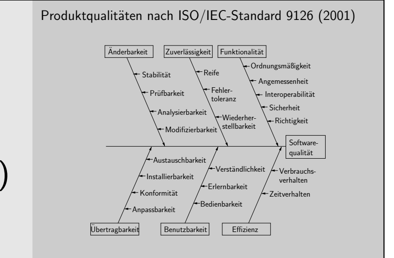
Im internationalen Standard ISO/IEC-Standard 9126 (2001) werden typische Softwarequalitäten in Form allgemeiner Kategorien und Subkategorien wie folgt aufgeführt:

Funktionalität: Vorhandensein von Funktionen mit festgelegten Eigenschaften, die die definierten Anforderungen erfüllen:

- Richtigkeit: Liefern der richtigen oder vereinbarten Ergebnisse oder Wirkungen, z.B. die benötigte Genauigkeit von berechneten Werten
- Angemessenheit: Eignung der Funktionen für spezifizierte Aufgaben, z.B. aufgabenorientierte Zusammensetzung von Funktionen aus Teilfunktionen
- Interoperabilität: Fähigkeit, mit vorgegebenen Systemen zusammen zu wirken
- Ordnungsmäßigkeit: Erfüllung von anwendungsspezifischen Normen, Vereinbarungen, gesetzlichen Bestimmungen und ähnlichen Vorschriften
- Sicherheit: Fähigkeit, unberechtigten Zugriff, sowohl versehentlich als auch vorsätzlich, auf Programme und Daten zu verhindern

Zuverlässigkeit: Fähigkeit der Software, ihr Leistungsniveau unter festgelegten Bedingungen über einen festgelegten Zeitraum zu bewahren:

- Reife: Geringe Versagenshäufigkeit durch Fehlerzustände
- Fehlertoleranz: Fähigkeit, ein spezifiziertes Leistungsniveau bei Softwarefehlern oder Nicht-Einhaltung ihrer spezifiziert Schnittstelle zu bewahren
- Wiederherstellbarkeit: Fähigkeit, bei einem Versagen das Leistungsniveau wiederherzustellen und die direkt betroffener Daten wiederzugewinnen



Benutzbarkeit: Aufwand, der zur Benutzung erforderlich ist, und individuelle Beurteilung der Benutzung durch eine festgelegte oder vorausgesetzte Benutzergruppe:

- Verständlichkeit: Aufwand für den Benutzer, das Konzept und die Anwendung zu verstehen
- Erlernbarkeit: Aufwand für den Benutzer, die Anwendung zu erlernen (z.B. Bedienung, Ein-, Ausgabe)
- Bedienbarkeit: Aufwand für den Benutzer, die Anwendung zu bedienen

Effizienz: Verhältnis zwischen dem Leistungsniveau der Software und dem Umfang der eingesetzten Betriebsmittel unter festgelegten Bedingungen:

- Zeitverhalten: Antwort- und Verarbeitungszeiten sowie Durchsatz bei der Funktionsausführung
- Verbrauchsverhalten: Anzahl und Dauer der benötigten Betriebsmittel für die Erfüllung der Funktionen

Änderbarkeit: Aufwand, der zur Durchführung vorgegebener Änderungen notwendig ist; Änderungen: Korrekturen, Verbesserungen oder Anpassungen an Änderungen der Umgebung, der Anforderungen und der funktionalen Spezifikationen:

- Analysierbarkeit: Aufwand, um Mängel oder Ursachen von Versagen zu diagnostizieren oder um änderungsbedürftige Teile zu bestimmen
- Modifizierbarkeit: Aufwand zur Ausführung von Verbesserungen, zur Fehlerbeseitigung oder Anpassung an Umgebungsänderungen
- Stabilität: Wahrscheinlichkeit des Auftretens unerwarteter Wirkungen von Änderungen
- Prüfbarkeit: Aufwand, der zur Prüfung der geänderten Software notwendig ist

Übertragbarkeit: Eignung der Software, von einer Umgebung in eine andere übertragen zu werden (Umgebung kann organisatorische Umgebung, Hardware oder Softwareumgebung einschließen):

- Anpassbarkeit: Software an verschiedene festgelegte Umgebungen anpassen
- Installierbarkeit: Aufwand, der zum Installieren der Software in einer festgelegten Umgebung notwendig ist
- Konformität: Grad, in dem die Software Normen oder Vereinbarungen zur Übertragbarkeit erfüllt
- Austauschbarkeit: Möglichkeit, diese Software anstelle einer spezifizierten anderen in der Umgebung jener Software zu verwenden, sowie der dafür notwendige Aufwand

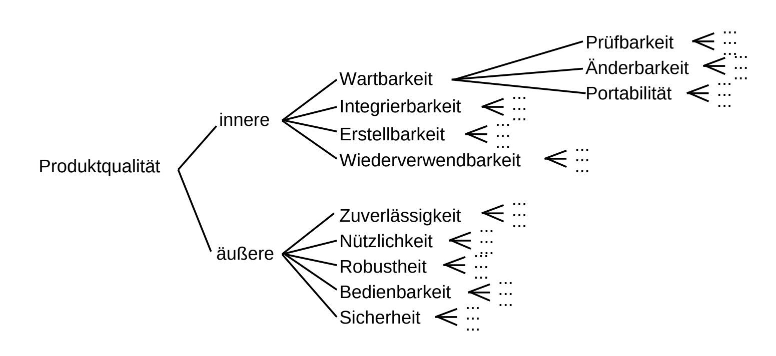
2008-11-04
Software-Projekt
└─Anforderungsanalyse
└─Anforderungsspezifikation
└─Produktqualitäten nach ISO/IEC-Standard 9126 (2001)



Diese Kategorien sind immer noch sehr allgemein und müssen für die Anforderungen an eine bestimmte Software konkretisiert und gewichtet werden. Dabei ist zu beachten, dass sich Qualitätsattribute gegenseitig (auch negativ) beeinflussen können. So stehen hohe Performanz und hohe Wartbarkeit meist in einem Spannungsfeld zueinander.

Die Konkretisierung und Gewichtung der Qualitätsattribute liefert das **Qualitätsmodell**. Für das Qualitätsmodell müssen dann Prüfkriterien festgelegt werden, anhand derer das Vorhandensein der gewünschten Eigenschaften überprüft werden kann. Im ISO/IEC-Standard 25000 (2005), der die Nachfolge von ISO/IEC-Standard 9126 (2001) angetreten hat, werden hierfür Metriken angegeben. Metriken sind ein numerisches Maß, das beschreibt, inwieweit eine bestimmte Qualität vorliegt. Durch Einsatz der Metriken wird die Qualität messbar. Über die wesentlichen Qualitätsattribute und sinnvolle Metriken zu ihrer Messung herrscht jedoch nicht immer Konsens.

Auch die Kategorisierung der Qualitätsattribute ist nicht immer gleich. Ludewig (2003) beispielsweise unterscheidet auf oberster Ebene nach inneren und äußeren Qualitäten. Äußere Qualitäten sind solche, die für den Benutzer direkt spürbar sind. Innere Qualitäten richten sich primär an den Entwickler, sind jedoch zumindest indirekt auch für den Benutzer spürbar durch den Aufwand und die Dauer für Änderungen an der Software, um sie an neue und geänderte Anforderungen anzupassen. Die folgende Grafik zeigt die Aufteilung nach Ludewig (2003):



Bedeutung einer Anforderungsspezifikation

Zweck	**Folge von Mängeln**
Abstimmung mit Kunden	Die Anforderungen bleiben ungeklärt, Wünsche des Kunden bleiben unberücksichtigt.
Entwurf	Entwerfer fehlt Vorgabe, darum mehr Kommunikation / eigene Vorstellung als Vorgabe.
Benutzerhandbuch	Basis für das Handbuch fehlt, es wird darum phänomenologisch verfasst.
Testvorbereitung	systematischer Test ist unmöglich
Abnahme	Korrektheit ist subjektiv, Streit ist unvermeidbar.
Wiederverwendung	nicht spezifizierte Systeme sind kaum durchschaubar, darum schwer wiederzuverwenden.
spätere Reimplementierung	Kompatibilität setzt voraus, dass man weiß, womit die neue Software kompatibel sein soll.

Bedeutung einer Anforderungsspezifikation

Zweck	Folge von Mängeln
Abstimmung mit Kunden	Die Anforderungen bleiben ungeklärt, Wünsche des Kunden bleiben unberücksichtigt.
Entwurf	Entwerfer fehlt Vorgabe, darum mehr Kommunikation / eigene Vorstellung als Vorgabe.
Benutzerhandbuch	Basis für das Handbuch fehlt, es wird darum phänomenologisch verfasst.
Testvorbereitung	systematischer Test ist unmöglich
Abnahme	Korrektheit ist subjektiv, Streit ist unvermeidbar.
Wiederverwendung	nicht spezifizierte Systeme sind kaum durchschaubar, darum schwer wiederzuverwenden.
spätere Reimplementierung	Kompatibilität setzt voraus, dass man weiß, womit die neue Software kompatibel sein soll.

Die Anforderungsspezifikation ist von zentraler Bedeutung für die Entwicklung. Sie wird von vielen Personen, die am Entwicklungsprozess beteiligt sind, als Grundlage ihrer Arbeit verwendet. Eine minderwertige Anforderungsspezifikation führt zwangsläufig zu Mängeln aller davon abhängiger weiterer Arbeitsprodukte. Da alle weiteren Arbeitspakete von der Anforderungsspezifikation abhängig sind, haben solche Mängel also eine nachhaltige Auswirkung auf die Qualität. Deshalb sollte man sich die Mühe machen, die Anforderungen genau zu beschreiben und sie am auch aufzuschreiben.

Man kann die Auswirkungen einer mangelhaften Anforderungsspezifikation unterscheiden anhand der Zwecke, die man mit der Anforderungsspezifikation verfolgt:

- **Abstimmung**: Die Anforderungsspezifikation dient der Abstimmung mit dem Kunden. Mängel in der Anforderungsspezifikation führen dazu, dass Anforderungen ungeklärt und Wünsche des Kunden unberücksichtigt bleiben. Ohne eine ausgearbeitete Anforderungsspezifikation kann der Kunde nicht ohne Weiteres in frühen Phasen de Projekts überprüfen, ob er richtig verstanden wurde und seine Wünsche wirklich umgesetzt werden.
- **Entwurf**: Der Softwarearchitekt entwirft auf Basis der Anforderungsspezifikation die Softwarearchitektur. Bei einer unzureichenden Anforderungsspezifikation fehlt dem Architekten dafür die notwendige Vorgabe. Dies führt zu einem erhöhten Bedarf an Kommunikation. Der Architekt muss beim Analytiker beziehungsweise beim Kunden nachfragen. Außerdem besteht die Gefahr, dass der Architekt seine eigene Vorstellung der Anforderungen entwickelt, die von der d Kunden abweichen könnte.
- **Benutzerhandbuch**: Anhand der Anforderungsspezifikation wird das Benutzerhandbuch geschrieben. Ohne Anforderungsspezifikation fehlt die Basis für das Handbuch. Dies führt dazu, dass es phänomenologisch erfasst wird; d der Autor des Benutzerhandbuchs schreibt auf, was er beim lauffähigen System beobachtet. Das ist oft unzureichend, weil er nicht alle Möglichkeiten kennt und damit beobachten kann.
- **Testvorbereitung**: Ohne präzise und vollständige Anforderungen ist ein systematischer Test unmöglich.
- **Vertrag und Abnahme**: In der Regel bildet die Anforderungsspezifikation die vertragliche Grundlage für den Auftrag. A Ende des Projekts wird bei der Abnahme überprüft, ob die Anforderungen erfüllt sind. Damit wird die Überprüfung de Korrektheit subjektiv und Streit ist unvermeidbar. Das führt zur Unzufriedenheit des Kunden und häufig zu Vertragsstreitigkeiten. Da aber die Anforderungen nicht vollständig und präzise beschrieben sind, ist ein Streit um das Recht mühsam.

Bedeutung einer Anforderungsspezifikation

Zweck	**Folge von Mängeln**
Abstimmung mit Kunden	Die Anforderungen bleiben ungeklärt, Wünsche des Kunden bleiben unberücksichtigt.
Entwurf	Entwerfer fehlt Vorgabe, darum mehr Kommunikation / eigene Vorstellung als Vorgabe.
Benutzerhandbuch	Basis für das Handbuch fehlt, es wird darum phänomenologisch verfasst.
Testvorbereitung	systematischer Test ist unmöglich
Abnahme	Korrektheit ist subjektiv, Streit ist unvermeidbar.
Wiederverwendung	nicht spezifizierte Systeme sind kaum durchschaubar, darum schwer wiederzuverwenden.
spätere Reimplementierung	Kompatibilität setzt voraus, dass man weiß, womit die neue Software kompatibel sein soll.

- **Wiederverwendung**: Ohne eine Beschreibung, was die Software leistet, kann man nicht ohne Weiteres beurteilen, ob man die Software wiederverwenden kann. Nicht spezifizierte Systeme sind kaum durchschaubar, darum schwer wiederzuverwenden.
- **Spätere Reimplementierung**: Soll das System später gegen ein neues System ersetzt werden, muss man wissen, was e tut. Kompatibilität setzt voraus, dass man weiß, womit die neue Software kompatibel sein soll.

Angestrebte Eigenschaften der Spezifikation

inhaltlich

(1) zutreffend (nicht „korrekt“!)
(2) vollständig (relativ zu den Wünschen des Kunden)
(3) widerspruchsfrei (oder konsistent, damit auch realisierbar)
(4) neutral d.h. abstrakt (und damit offen für beliebigen Entwurf)

in der Darstellung

(5) leicht verständlich (für alle Zielgruppen!)
(6) präzise (schließt Umgangssprache aus)

in der Form

(7) leicht erstellbar (was die Notationen und Modelle betrifft)
(8) leicht verwaltbar (also auch zweckmäßig strukturiert)
(9) objektivierbar (auch – nicht sinnvoll – „testbar“ genannt)

Diese Merkmale konkurrieren, d.h. die Erfüllung des einen erschwert oder verhindert die Erfüllung des anderen.

Angestrebte Eigenschaften der Spezifikation

inhaltlich
(1) zutreffend (nicht „korrekt"!)
(2) vollständig (relativ zu den Wünschen des Kunden)
(3) widerspruchsfrei (oder konsistent, damit auch realisierbar)
(4) neutral d.h. abstrakt (und damit offen für beliebigen Entwurf)
in der Darstellung
(5) leicht verständlich (für alle Zielgruppen!)
(6) präzise (schließt Umgangssprache aus)
in der Form
(7) leicht erstellbar (was die Notationen und Modelle betrifft)
(8) leicht verwaltbar (also auch zweckmäßig strukturiert)
(9) objektivierbar (auch – nicht sinnvoll – „testbar" genannt)

Diese Merkmale konkurrieren, d.h. die Erfüllung des einen erschwert oder verhindert die Erfüllung des anderen.

Weil eine Anforderungsspezifikation von so tragender Bedeutung ist, streben wir eine hochqualitative an. Welche Eigenschaften müssen wir aber hierzu anstreben? Die anzustrebenden Eigenschaften lassen sich wie folgt kategorisieren:

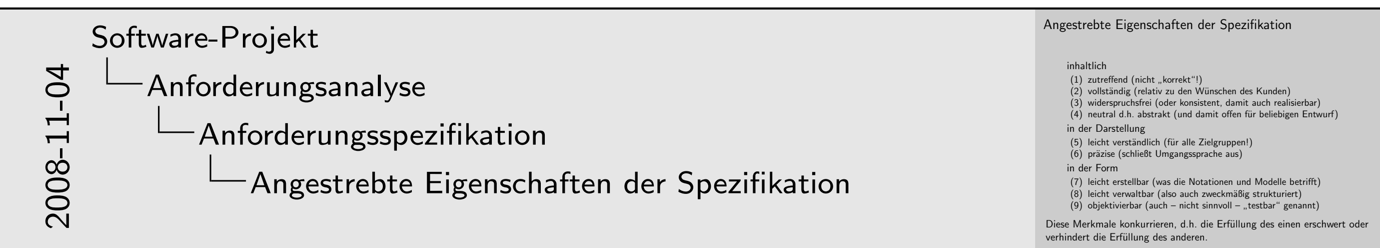
Den **Inhalt** betreffend:

(1) **zutreffend**: die Anforderungen des Kunden werden richtig wieder gegeben; an manchen Stellen wird für *zutreffend* auc der Begriff *korrekt* synonym verwendet; wir werden aber im Folgenden den Begriff *korrekt* für das Verhältnis der Implementierung zur Anforderungsspezifikation reservieren; eine Implementierung ist dann korrekt, wenn sie die spezifizierten Anforderungen richtig erfüllt. Daraus kann man jedoch noch nicht unmittelbar schließen, dass die Implementierung auch das implementiert, was der Kunde tatsächlich will. Hierzu ist es noch notwendig, dass die Anforderungsspezifikation auch die Wünsche des Kunden zutreffend wiedergibt.

(2) **vollständig**: alle Wünsche des Kunden werden berücksichtigt.

(3) **widerspruchsfrei** (oder konsistent, damit auch realisierbar): die Anforderungen widersprechen sich nicht; würden sie sic widersprechen, könnte man nichts implementieren, was alle Anforderungen erfüllt.

(4) **neutral**, d.h. abstrakt: die Anforderungen sind aus Sicht des Kunden formuliert und nehmen keine unnötigen technisch Details vorweg. Implementierungsaspekte gehören nicht in die Anforderungsspezifikation, weil sie den Kunden in der Regel nicht interessieren, sondern eher verwirren. Außerdem bleibt damit dem Architekten beim Entwurf ein größtmöglicher Gestaltungsspielraum.


2008-11-04
Software-Projekt
└─Anforderungsanalyse
└─Anforderungsspezifikation
└─Angestrebte Eigenschaften der Spezifikation

Angestrebte Eigenschaften der Spezifikation

inhaltlich
(1) zutreffend (nicht „korrekt"!)
(2) vollständig (relativ zu den Wünschen des Kunden)
(3) widerspruchsfrei (oder konsistent, damit auch realisierbar)
(4) neutral d.h. abstrakt (und damit offen für beliebigen Entwurf)
in der Darstellung
(5) leicht verständlich (für alle Zielgruppen!)
(6) präzise (schließt Umgangssprache aus)
in der Form
(7) leicht erstellbar (was die Notationen und Modelle betrifft)
(8) leicht verwaltbar (also auch zweckmäßig strukturiert)
(9) objektivierbar (auch – nicht sinnvoll – „testbar" genannt)

Diese Merkmale konkurrieren, d.h. die Erfüllung des einen erschwert oder verhindert die Erfüllung des anderen.

Die **Darstellung** betreffend:

(5) **leicht verständlich**: Die Anforderungsspezifikation ist leicht verständlich für alle Zielgruppen. Dazu gehören neben Entwicklern mit technischem Verständnis auch Kunden und Benutzer, die meist mit formalen Methoden und Notation nichts anzufangen wissen. Selbst die Verwendung der *Unified Modeling Language* ist für einen Kunden und Benutzer nicht ohne Weiteres zumutbar, auch wenn manche sie als Notation für die Kommunikation mit Kunden und Benutzern anpreisen. Die Vorteile einer (semi-)formalen Notation ist ihre Präzision und Prägnanz. Aus diesem Grund würde man sich ihren Einsatz wünschen. Wird eine solche Notation verwendet, muss der Kunde aber darin geschult werden – ode würden wir einen Vertrag in Hieroglyphen unterzeichnen?

(6) **präzise**: die Anforderungen müssen unmissverständlich formuliert sein, sonst besteht die Gefahr, dass etwas Falsches implementiert wird. Dies schließt Umgangssprache, die sich durch vage Begriffe und Mehrdeutigkeiten auszeichnet, au Ebenso sollten sprachliche Weichmacher wie *könnte*, *müsste*, *circa*, *ungefähr* etc. vermieden werden.

Angestrebte Eigenschaften der Spezifikation

inhaltlich
(1) zutreffend (nicht „korrekt“!)
(2) vollständig (relativ zu den Wünschen des Kunden)
(3) widerspruchsfrei (oder konsistent, damit auch realisierbar)
(4) neutral d.h. abstrakt (und damit offen für beliebigen Entwurf)

in der Darstellung
(5) leicht verständlich (für alle Zielgruppen!)
(6) präzise (schließt Umgangssprache aus)

in der Form
(7) leicht erstellbar (was die Notationen und Modelle betrifft)
(8) leicht verwaltbar (also auch zweckmäßig strukturiert)
(9) objektivierbar (auch – nicht sinnvoll – „testbar“ genannt)

Diese Merkmale konkurrieren, d.h. die Erfüllung des einen erschwert oder verhindert die Erfüllung des anderen.

Die **Form** betreffend:

(7) **leicht erstellbar**: die Anforderungsspezifikation ist in vielen Fällen ein sehr umfangreiches Dokument. Darum ist es wichtig, das es leicht erstellt werden kann. Somit ist die Auswahl geeigneter Notationen und Modelle bedeutsam. Häu wird sie nicht nur von einem Autoren erstellt, so dass die Zusammenarbeit mehrerer Autoren unterstützt werden muss Die Anforderungsspezifikation muss also modular aufgebaut sein.

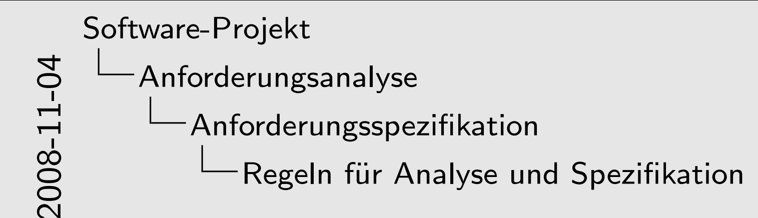
(8) **leicht verwaltbar**: die Anforderungsspezifikation muss durch zweckmäßige Strukturierung, eindeutige Referenzierbarkei und explizite Querbezüge leicht überarbeitet und versioniert werden können. Idealerweise ist der Code über den Entwu hin zu den Anforderungen verknüpft, um eine hohe Nachvollziehbarkeit zwischen diesen Dokumenten zu gewährleisten da die Dokumente nachgeführt werden müssen, wenn sich Dinge ändern. Hierfür wurden eine Reihe von Werkzeugen, wie z.B. *Doors*, entwickelt

(9) **objektivierbar**: für jede Anforderung muss es möglich sein, eine Prüfprozedur anzugeben, mit deren Hilfe man objektiv entscheiden kann, ob das resultierende System die Anforderung erfüllt; synonym zu *objektivierbar* wird häufig auch de Begriff *testbar* verwendet; dieser Begriff suggeriert jedoch, dass man das System für die Prüfung ausführen können mu (vgl. das Kursmodul *Test*); die meisten Prüfungen lassen sich aber ohne Ausführung des Systems durchführen.
Die Anforderung „Die Programm soll eine hohe Performanz aufweisen“ ist ein Negativbeispiel für eine objektivierbare Anforderung. Hier ist unklar, was „hohe Performanz“ genau bedeutet. Statt dessen sollte die Anforderung vielmehr in der folgenden Weise formuliert werden: „Die Ausgabe soll in mindestens 60 % aller Fälle nach spätestens 20 Sekunden erfolgen; nach höchstens 30 Sekunden soll die Ausgabe in 100 % aller Fälle erfolgen. Die Berechnung muss mit maxim 5 MB Hauptspeicher und 100 MB Plattenplatz auskommen“.

Leider ist es nicht immer möglich, alle diese Merkmale bei einer Anforderungsspezifikation zu erfüllen, weil die Merkmale konkurrieren können, das heißt die Erfüllung des einen erschwert oder verhindert die Erfüllung des anderen. Präzise Anforderungen sind in der Regel nur mit Hilfe formaler Spezifikationsprachen zu erreichen, die sind aber für einen Kunden meist nicht verständlich.

Regeln für Analyse und Spezifikation

- Ein Begriffslexikon anlegen und entwickeln
- Von der Aufgabe ausgehen, nicht von ihrer Lösung
- Daten suchen, nicht Programmabläufe beschreiben
- Abstraktionsebene nicht in einer Darstellung wechseln
- Die Spezifikation nach Aspekten organisieren
- Ein Mengengerüst bilden
- Den Kunden (Benutzer) einbeziehen
- Geeignete Sprachen und Werkzeuge verwenden
- Die Spezifikation so früh wie möglich prüfen und dem Konfigurationsmanagement unterstellen
- Die Spezifikation intensiv verwenden



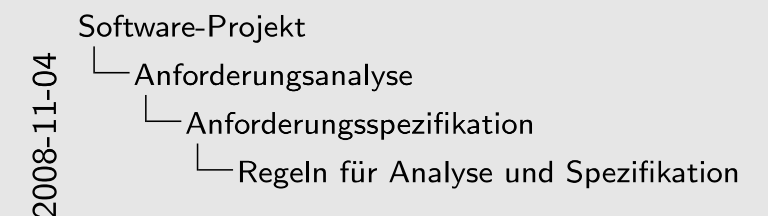
Regeln für Analyse und Spezifikation

- Ein Begriffslexikon anlegen und entwickeln
- Von der Aufgabe ausgehen, nicht von ihrer Lösung
- Daten suchen, nicht Programmabläufe beschreiben
- Abstraktionsebene nicht in einer Darstellung wechseln
- Die Spezifikation nach Aspekten organisieren
- Ein Mengengerüst bilden
- Den Kunden (Benutzer) einbeziehen
- Geeignete Sprachen und Werkzeuge verwenden
- Die Spezifikation so früh wie möglich prüfen und dem Konfigurationsmanagement unterstellen
- Die Spezifikation intensiv verwenden

Um eine gute Anforderungsspezifikation zu erreichen, ist es ratsam, einige Regeln zu befolgen.

- Ein Begriffslexikon (auch: Glossar) anlegen und entwickeln. Das Begriffslexikon beschreibt alle Begriffe, die der Kunde und wir benützen, die nicht unmittelbar offensichtlich sind. Die schriftliche Niederlegung der Bedeutung aller Begriffe zwingt uns und den Kunden zur notwendigen Präzision in unserem Dialog und hilft damit, Missverständnisse zu vermeiden. Hier sollten auch alle Synonyme aufgezählt werden. Im eigentlichen Inhalt der schriftlichen Anforderungsspezifikation sollten wir aber Synonyme vermeiden und immer nur dasselbe Wort benutzen. Die Anforderungsspezifikation ist kein literatisches Werk, das durch hohe Sprachvariation gefallen will, sondern soll ein präzises, leicht verständliches technisches Dokument sein. Unterschiedliche Worte für denselben Sachverhalten verwirr nur.
- Von der Aufgabe ausgehen, nicht von ihrer Lösung. Die Anforderungsspezifkation beschreibt das zu implementierende System aus Sicht des Kunden. Wie das System intern implementiert wird, interessiert den Kunden nicht, sondern verwirrt ihn eher, weil er sich mit Implementierungsfragen in der Regel nicht auskennt. Implementierungsfragen gehöre in ein Entwurfsdokument, wie z.B. die Architekturbeschreibung. Wir sollten den Gestaltungsraum des Softwarearchitekten durch die Anforderungsspezifikation nicht unnötig einschränken.
- Daten suchen, nicht Programmabläufe beschreiben. Primär sind wir an den Daten, d.h. den Objekten, der Anwendungsdomäne interessiert. Sie ändern sich meist weniger als die Abläufe, in denen sie verarbeitet werden. Natürlich gehören zur Beschreibung in der Anforderungsspezifikation die Kommunikation zwischen den Objekten und welche Operationen sie prinzipiell beherrschen. Die Software muss ja Arbeitsflüsse abbilden. Wie aber diese Operation intern implementiert werden (der Programmablauf) ist ein Detail der Implementierung und sollte erst später im Entwu oder in der Implementierungsphase spezifiziert werden.
- Abstraktionsebene nicht in einer Darstellung wechseln. Um den Leser zu führen statt ihn zu verwirren, sollten wir uns Details für vertiefende Abschnitte aufheben. Die Anforderungsspezifikation sollte so gestaltet sein, dass der Leser erst einen allgemeinen Überblick über alle Anforderungen an das System erhält und erst dann einen detaillierten Einblick i die einzelnen Anforderungen. Ein Wechsel zwischen diesen Ebenen verwirrt unnötig.

2008-11-04

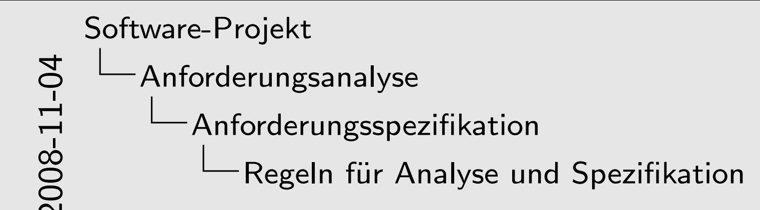


Regeln für Analyse und Spezifikation

- Ein Begriffslexikon anlegen und entwickeln
- Von der Aufgabe ausgehen, nicht von ihrer Lösung
- Daten suchen, nicht Programmabläufe beschreiben
- Abstraktionsebene nicht in einer Darstellung wechseln
- Die Spezifikation nach Aspekten organisieren
- Ein Mengengerüst bilden
- Den Kunden (Benutzer) einbeziehen
- Geeignete Sprachen und Werkzeuge verwenden
- Die Spezifikation so früh wie möglich prüfen und dem Konfigurationsmanagement unterstellen
- Die Spezifikation intensiv verwenden

- Die Spezifikation nach Aspekten organisieren. Die Spezifikation ist nicht selten ein Dokument mit mehreren hundert Seiten. Es kann nicht ohne Unterbrechung gelesen werden. Außerdem wird es von Lesern mit sehr unterschiedlichen Interessen gelesen. Dazu gehören auf Kundenseite neben dem Auftraggeber (auch hier gibt es meist verschiedene Rolle der Auftraggeber, der für den Inhalt verantwortlich ist und der Auftraggeber, der für die Finanzierung und Termintreu verantwortlich ist) die verschiedenen Benutzerklassen. Auf der Seite der Hersteller wird das Dokument von den Architekten, den Programmierern, den Handbuchautoren und den Testern gelesen. Alle interessieren sich für unterschiedliche Belange. Um die vielen Arten von Lesern zu unterstützen, muss die Spezifikation uniform nach relevanten Aspekten organisiert werden.
- Ein Mengengerüst bilden. Damit der Architekt eine Vorstellung über die Anforderungen an die Belastbarkeit des Syste erhält, um das System dafür entsprechend auszulegen, muss er wissen, mit wie vielen Daten er es zu tun haben wird. macht einen gravierenden Unterschied, ob ein System etwa 10 oder 10.000 Benutzer unterstützen muss. Damit der Architekt das System auch für zukünftige Anforderungen skalierbar entwerfen kann, sollten neben dem heutigen Mengengerüst auch realistische Prognosen gemacht werden, wie sich das Mengengerüst in Zukunft entwickeln wird.
- Den Kunden (Benutzer) einbeziehen. Wir entwickeln für den Kunden, der uns dafür bezahlt, und für die Benutzer, die unser System später benutzen sollen. Ihre Belange sind Ausgangspunkt unseres Projekts. Sie können an verschiedenen Stellen im Entwicklungsprozess eingebunden werden, nicht nur während der Anforderungsanalyse und am Ende beim Akzeptanztest. Beim partizipativen Vorgehen beispielsweise werden wir in kurzen Zeitabständen Prototypen und verschiedene Inkremente des Systems vorführen, um ihr Feedback einzuholen. Ganz besonders benötigen wir sie aber während der initialen Anforderungsanalyse. Um sicher zu stellen, dass wir die Anforderungsspezifikation die Wünsche d Kunden und der Benutzer treffend widerspiegelt, sollte die Anforderungsspezifikation in einem Review mit Kunden un Benutzern abgenommen werden.

2008-11-04

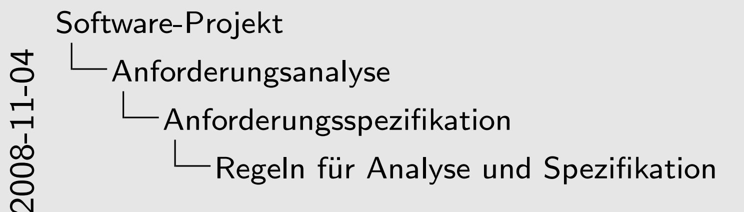


Regeln für Analyse und Spezifikation

- Ein Begriffslexikon anlegen und entwickeln
- Von der Aufgabe ausgehen, nicht von ihrer Lösung
- Daten suchen, nicht Programmabläufe beschreiben
- Abstraktionsebene nicht in einer Darstellung wechseln
- Die Spezifikation nach Aspekten organisieren
- Ein Mengengerüst bilden
- Den Kunden (Benutzer) einbeziehen
- Geeignete Sprachen und Werkzeuge verwenden
- Die Spezifikation so früh wie möglich prüfen und dem Konfigurationsmanagement unterstellen
- Die Spezifikation intensiv verwenden

- Geeignete Sprachen und Werkzeuge verwenden. Die Anforderungsspezifikation sollte möglichst präzise, aber eben auch verständlich sein. Hierzu kann man Diagrammen benutzen. Die UML beispielsweise enthält eine Reihe von Diagrammtypen, mit deren Hilfe man bestimmte Aspekte verständlich und genau beschreiben kann. Für das Datenmodell eignen sich zum Beispiel Klassendiagramme. Es versteht sich von selbst, dass wir dem Kunden unsere Notation erklären müssen. Zur Erstellung des Dokuments bedient man sich am besten spezieller Werkzeuge. Neben herkömmlicher Editierfunktionen können uns hierbei Werkzeuge helfen, Anforderungen eindeutig zu nummerieren sowi Verweise einzufügen, zu verfolgen und zu prüfen. Diese Werkzeuge können uns auch einen festen Rahmen vorgeben. Einfache Prüfungen auf Vollständigkeit werden damit möglich.
- Die Spezifikation so früh wie möglich prüfen und dem Konfigurationsmanagement unterstellen. Die Anforderungsspezifikation sollte wir selbst mit internen Reviews überprüfen. Nach Fertigstellung der Anforderungsspezifikation sollte mindestens eine Prüfung mit dem Kunden und den Benutzern stattfinden. Weitere no frühere Prüfungen kann man mit Hilfe von Prototypen erreichen.
  Die Anforderungsspezifikation wird sich ändern, weil in Prüfungen und auch später Fehler und Lücken gefunden werde Außerdem können sich bei einer längeren Projektlaufzeit Anforderungen auch noch während der Entwicklung ändern aufgrund geänderter Rahmenbedingungen. Zudem arbeiten meist mehrere Personen an der Anforderungsspezifikation. Aus diesem Grund muss die Anforderungsspezifikation von Anbeginn unter das Konfigurationsmanagement gestellt werden.
- Die Spezifikation intensiv verwenden. Die Anforderungsspezifikation ist der Ausgang für alle weiteren Entwicklungen. S wird nicht nur benutzt beim Vertragsabschluss von Auftragnehmer und -geber sondern für eine Reihe weiterer Persone die für die Entwicklung verantwortlich sind. Dazu gehören Architekten, Programmierer, Tester und Handbuchautoren. sollte eine Selbstverständlichkeit sein, dass diese Menschen die Spezifikation intensiv nutzen. Leider hat die Praxis gezeigt, dass z.B. viele Programmierer Fehler deshalb machen, weil sie die Anforderungsspezifikation niemals gelesen haben.

2008-11-04



Regeln für Analyse und Spezifikation

- Ein Begriffslexikon anlegen und entwickeln
- Von der Aufgabe ausgehen, nicht von ihrer Lösung
- Daten suchen, nicht Programmabläufe beschreiben
- Abstraktionsebene nicht in einer Darstellung wechseln
- Die Spezifikation nach Aspekten organisieren
- Ein Mengengerüst bilden
- Den Kunden (Benutzer) einbeziehen
- Geeignete Sprachen und Werkzeuge verwenden
- Die Spezifikation so früh wie möglich prüfen und dem Konfigurationsmanagement unterstellen
- Die Spezifikation intensiv verwenden

Nachdem wir wissen, welche Qualitäten der Anforderungsspezifikation wir anstreben und welche Regeln wir einhalten sollen, werden wir uns nun mit ihrem Inhalt und ihrer Struktur auseinander setzen. Glücklicherweise haben sich dazu bereits viele Spezialisten Gedanken gemacht, die sogar in einen Standard gemündet sind. Da wir das Rad nicht neu erfinden wollen und die Orientierung an Standards eine gute Ingenieurstugend ist, nehmen wir uns diesen Standard zum Vorbild. Wir können diesen Standard als einen Rahmen für unsere Spezifikation verwenden und laufen so weniger Gefahr, etwas zu vergessen.

Es handelt sich dabei um den *IEEE Recommended Practice for Software Requirements Specifications, IEEE Standard 830.1998*. Er setzt vieles von dem um, was wir bislang kennen gelernt haben. Wir werden hierzu Konkretisierungen und darüber hinaus führende Ergänzungen kennen lernen (Ergebnisse der Soll-Analyse, Datenmodell und Prognosen für zukünftige Anforderungen), die auch beschrieben sein sollten.

Inhalt der Anforderungsspezifikation nach IEEE Standard 830.1998

1. Einführung
   1.1 Zweck
   1.2 Rahmen 3889
   1.3 Definitionen, Akronyme und Abkürzungen
   1.4 Referenzen
   1.5 Übersicht über das Dokument
2. Allgemeine Beschreibung
   2.1 Produktperspektive
   2.2 Produktfunktionen
   2.3 Charakteristika der Benutzer
   2.4 Einschränkungen
   2.5 Annahmen und Abhängigkeiten
3. Detaillierte Beschreibung

Inhalt der Anforderungsspezifikation nach IEEE Standard 830.1998

1. Einführung
   1.1 Zweck
   1.2 Rahmen 3889
   1.3 Definitionen, Akronyme und Abkürzungen
   1.4 Referenzen
   1.5 Übersicht über das Dokument
2. Allgemeine Beschreibung
   2.1 Produktperspektive
   2.2 Produktfunktionen
   2.3 Charakteristika der Benutzer
   2.4 Einschränkungen
   2.5 Annahmen und Abhängigkeiten
3. Detaillierte Beschreibung

Der Standard gibt eine dreiteilige Struktur vor: eine Einführung, eine allgemeine Beschreibung und eine detaillierte Beschreibung. Er folgt hierin den oben erwähnten Regeln, nach Aspekten und Detaillierungsgrad zu ordnen. Die Struktur bis in die zweite Gliederungsebene ist wie folgt:

1. Einführung
   1.1 Zweck
   1.2 Rahmen
   1.3 Definitionen, Akronyme und Abkürzungen
   1.4 Referenzen
   1.5 Übersicht über das Dokument
2. Allgemeine Beschreibung
   2.1 Produktperspektive
   2.2 Produktfunktionen
   2.3 Charakteristika der Benutzer
   2.4 Einschränkungen
   2.5 Annahmen und Abhängigkeiten
3. Detaillierte Beschreibung

Wir werden diese Abschnitte nun im Einzelnen im Detail kennen lernen. Ein Beispiel für eine ausführliche Anforderungsspezifikation finden Sie unter . . . <<**hier ergänzen: gute Spec. vom ersten SWP zu PDA hinzufügen**>>

Inhalt der Anforderungsspezifikation nach IEEE Standard 830.1998

1. Einführung

1.1 Zweck

- Zweck der Spezifikation
- adressierte Leser

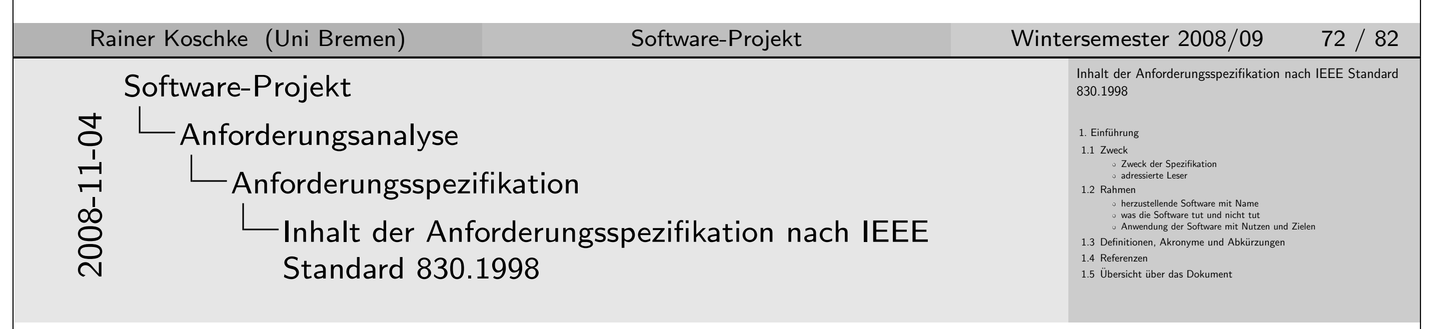
1.2 Rahmen

- herzustellende Software mit Name
- was die Software tut und nicht tut
- Anwendung der Software mit Nutzen und Zielen

1.3 Definitionen, Akronyme und Abkürzungen

1.4 Referenzen

1.5 Übersicht über das Dokument



Das Ziel der Einführung ist es, den Kontext und Rahmen der Anforderungsspezifikation abzustecken. Sie hat folgende Struktur:

1.1 **Zweck**
Dieser Abschnitt beschreibt den Zweck der Spezifikation und an welche Leser sie sich wendet. Meist geschieht dies durch eine kurze Beschreibung des Projektkontextes und der Ziele des Projekts. Dann folgt der Hinweis, dass in dieser Dokument die Anforderungen dieses System spezifiziert werden sollen und oft die Aussage, dass dieses Dokument die vertragliche Grundlage bildet. Für die adressierten Leser werden mindestens die konkreten Namen der Kunden und Vertreter möglicher Benutzer genannt.

1.2 **Rahmen**
Hier geben wir der zu erstellenden Software einen Namen und beschreiben ganz kurz, was die herzustellende Software leisten soll und auch, was wir explizit nicht von ihr erwarten, um gleich zu Anfang falschen Vorstellungen Einhalt zu gebieten. Es folgt die Beschreibung der Anwendung der Software mit ihrem Nutzen und ihren Zielen. Dieser Abschnitt soll nur einen ersten Überblick vermitteln, was die Software genau machen soll, folgt in den zwei nachfolgenden Kapite
Bsp.: *Der PDA-basierte Einkaufsassistent, den wir als laufendes Beispiel verwenden, soll den Namen PDAss tragen.*

1.3 **Definitionen, Akronyme und Abkürzungen**
Hier ist er der Ort, an dem Begriffe definiert und Akronyme und Abkürzungen eingeführt werden können, die im weiteren Dokument benutzt werden. Damit wird der Standard unserer oben geäußerten Forderung gerecht, ein Begriffslexikon (Glossar) zum Ausschluss von Missverständnissen zu verwenden.
Bsp.: *Wir würden hier beispielsweise das Akronym PDA für „Personal Digital Assistant" erläutern.*

1.4 **Referenzen**
Hier können weitere Dokumente aufgeführt werden, die für das Verständnis der Anforderungsspezifikation hilfreich sin und im folgenden Text referenziert werden. Dazu sollte auch z.B. der IEEE-Standard für diese Anforderungsspezifikati gehören.
Bsp.: *Hier könnten wir die Fahrradherstellerkataloge und weitere Dokumente, die wir in der Ist-Analyse gefunden habe nennen.*

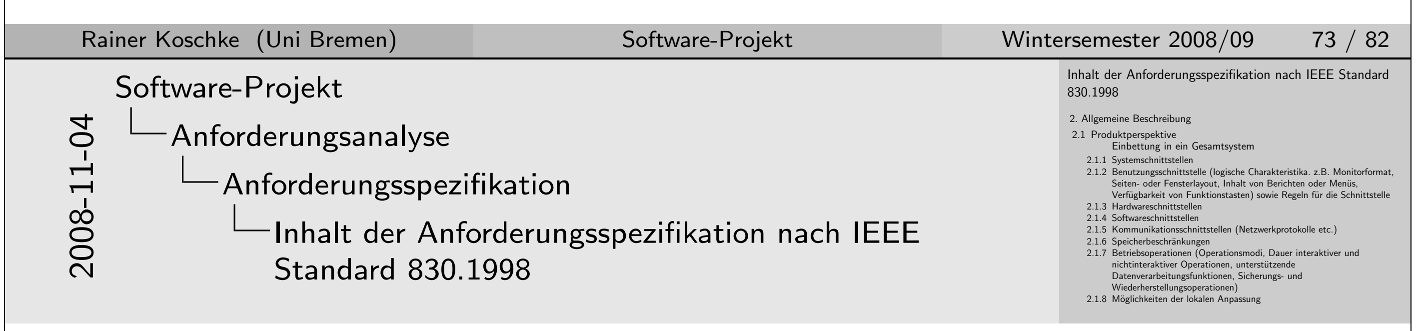
1.5 **Übersicht über das Dokument**
Schließlich erfolgt hier in diesem Abschnitt eine kurze Übersicht für den Leser über die weiteren Kapitel und Abschnitt dieser Anforderungsspezfikation.

Inhalt der Anforderungsspezifikation nach IEEE Standard 830.1998

2. Allgemeine Beschreibung

2.1 Produktperspektive
Einbettung in ein Gesamtsystem

2.1.1 Systemschnittstellen
2.1.2 Benutzungsschnittstelle (logische Charakteristika. z.B. Monitorformat, Seiten- oder Fensterlayout, Inhalt von Berichten oder Menüs, Verfügbarkeit von Funktionstasten) sowie Regeln für die Schnittstelle
2.1.3 Hardwareschnittstellen
2.1.4 Softwareschnittstellen
2.1.5 Kommunikationsschnittstellen (Netzwerkprotokolle etc.)
2.1.6 Speicherbeschränkungen
2.1.7 Betriebsoperationen (Operationsmodi, Dauer interaktiver und nichtinteraktiver Operationen, unterstützende Datenverarbeitungsfunktionen, Sicherungs- und Wiederherstellungsoperationen)
2.1.8 Möglichkeiten der lokalen Anpassung



Das Kapitel 2 beschreibt Allgemeines sowie die Anforderungen im Einzelnen, ohne jedoch gleich alle möglichen Details zu nennen (letztere folgen im dritten Kapitel).

2.1 **Produktperspektive**

Dieser Abschnitt bettet das zu implementierende System in ein Gesamtsystem ein. Dazu werden alle seine Schnittstellen zu anderen Systemen und zum Benutzer beschrieben sowie weitere globale Einschränkungen.

Wir werden bei der Beschreibung der Schnittstellen feststellen, dass es nicht immer einfach ist, eine Schnittstelle einer der unten aufgeführten Kategorien eindeutig zuzuordnen. Im Zweifel entscheiden wir uns für eine und fügen in den anderen Abschnitten einen Querbezug zu dieser Beschreibung ein. Primär ist wichtig, dass die Schnittstelle beschrieben ist. Durch die Querbezüge stellen wir sicher, dass wir diese Beschreibung finden werden, egal in welchem Abschnitt wir intuitiv suchen.

2.1.1 **Systemschnittstellen**
Über Systemschnittstellen ist das zu implementierende System mit anderen Systemen verbunden, mit denen es zusammen arbeiten soll. Dazu gehören z.B. Import- und Exportschnittstellen für den Datenaustausch, Scripting-Schnittstellen, über die sich das System programmieren lässt, sowie Application Interfaces (API), über das man das System programmatisch steuern kann.
Bsp.: *Hier könnten wir für PDAss fordern, dass man die Herstellerkataloge in einem zu definierenden elektronischen XML-basierten Format einlesen können soll.*

2.1.2 Benutzungsschnittstelle

Eine besondere Schnittstelle ist die, die dem Benutzer angeboten wird. Die detaillierte Beschreibung dieser Schnittstel erfolgt in Abschnitt 3.1.1. Hier werden lediglich die logischen Charakteristika der Schnittstelle erläutert, z.B. das Monitorformat, das Seiten- oder Fensterlayout, den Inhalt von Berichten oder Menüs oder die Verfügbarkeit von Funktionstasten. Außerdem werden alle relevanten Regeln für das Design der Schnittstelle genannt, wie z.B. die allgemeinen (in aller Regel ohnehin verbindlichen) Richtlinien der Norm EN ISO 9241-10:1995 (1995) für die Gebrauchstauglichkeit von Software. Die Angaben müssen aber hinreichend konkretisiert werden. Diese Regeln können auch eine einfache Liste von "Dos and Don'ts" sein, wie sich die Schnittstelle dem Benutzer darstellen soll. Beispielsweise ob kurze oder lange Fehlermeldungen vorzuziehen sind. In jedem Fall gilt auch hier, dass die Anforderungen objektivierbar sein müssen. Statt „benutzerfreundlich" sollte es also besser „Persona X kann die Funkti *Sortiment einsehen* in zwei Minuten ausführen (nach einer Schulung von zehn Minuten)" heißen.

Charakteristika der Benutzer werden in Abschnitt 2.3 angegeben.

Bsp.: *Hier würden wir z.B. die Auflösung des PDA-Monitors für PDAss, die Farbtiefe, die Eingabemöglichkeiten über Stift und nur wenige Tasten und die minimalen Schriftgrößen etc. angeben.*

2.1.3 Hardwareschnittstellen

Hardwareschnittstellen sind Schnittstellen zu Hardware, die das System voraussetzt beziehungsweise bedient. Dieser Abschnitt ist insbesondere für eingebettete Systeme von großer Bedeutung. Für alle Arten von Systemen würde man hier aber die zu unterstützenden Prozessortypen beschreiben, auf denen die Software später laufen soll. Hardware, die eher der Benutzungsschnittstelle zuzuordnen ist, wird in Abschnitt 2.1.2 beschrieben.



2008-11-04
Software-Projekt
Anforderungsanalyse
Anforderungsspezifikation
Inhalt der Anforderungsspezifikation nach IEEE Standard 830.1998

Inhalt der Anforderungsspezifikation nach IEEE Standard 830.1998

2. Allgemeine Beschreibung
2.1 Produktperspektive
Einbettung in ein Gesamtsystem
2.1.1 Systemschnittstellen
2.1.2 Benutzungsschnittstelle (logische Charakteristika. z.B. Monitorformat, Seiten- oder Fensterlayout, Inhalt von Berichten oder Menüs, Verfügbarkeit von Funktionstasten) sowie Regeln für die Schnittstelle
2.1.3 Hardwareschnittstellen
2.1.4 Softwareschnittstellen
2.1.5 Kommunikationsschnittstellen (Netzwerkprotokolle etc.)
2.1.6 Speicherbeschränkungen
2.1.7 Betriebsoperationen (Operationsmodi, Dauer interaktiver und nichtinteraktiver Operationen, unterstützende Datenverarbeitungsfunktionen, Sicherungs- und Wiederherstellungsoperationen)
2.1.8 Möglichkeiten der lokalen Anpassung

2.1.4 Softwareschnittstellen

Softwareschnittstellen sind Schnittstellen zu Software, die Bestandteil des System werden soll. Dazu gehören Betriebssysteme, höhere Dienste wie z.B. Datenbanken sowie wiederverwendbare Bibliotheken, die eingebunden werde sollen.

Wir erinnern uns, dass Implementierungsdetails eigentlich nicht in die Anforderungsspezifikation gehören. Insofern dürf wir nur solche Softwareschnittstellen hier aufzählen, die tatsächlich von Interesse für den Kunden sind, z.B. dann wenn er diese Software bereits einsetzt und in diesem Kontext wieder verwenden möchte bzw. wenn sich aus ihrer Verwendu weitere lizenzrechtliche Aspekte für den Kunden ergeben würden.

Bsp.: *Der Ladenverkäufer hat eine MySQL-Datenbank der Version 4.1 bereits im Einsatz und besteht darauf, dass unsere Software alle persistenten Daten darin speichern soll.*

2.1.5 Kommunikationsschnittstellen

Sofern das System mit anderen Systemen über ein Netzwerk kommuniziert, beschreiben wir hierfür die Kommunikationsschnittstelle sowie Netzwerkprotokolle etc. Auch hier gilt wieder, dass nur solche Schnittstellen interessieren, die zu anderen Systemen führen beziehungsweise für die der Kunde sorgen muss, damit unser System arbeiten kann. Ist eine solche Schnittstelle vollständig intern und wird von uns mitgeliefert, handelt es sich um ein Implementierungsdetail, das wir nicht in der Anforderungsspezifikation nennen.

Bsp.: *Wir wollen den PDA mit dem Ladenrechner über WLAN kommunizieren lassen. Wir setzen voraus, dass der Kun für das WLAN in seinem Laden sorgt. Dann wäre WLAN mit einem Netzwerkprotokoll wie TCP/IP in der Anforderungsspezifikation zu erwähnen. Wir dürfen dann in unserer Implementierung voraussetzen, dass die Ausführungsumgebung diese Komponenten bereit hält.*

2.1.6 Speicherbeschränkungen

Hier nennen wir die mindestens verfügbare Größe des Hauptspeichers und der Festplatten, die das korrekte Funktionier unserer Software voraussetzen darf. Anforderungen an die Laufzeit werden im Kontext der Operationen festgelegt.

Inhalt der Anforderungsspezifikation nach IEEE Standard 830.1998

2. Allgemeine Beschreibung

2.1 Produktperspektive
Einbettung in ein Gesamtsystem
- 2.1.1 Systemschnittstellen
- 2.1.2 Benutzungsschnittstelle (logische Charakteristika. z.B. Monitorformat, Seiten- oder Fensterlayout, Inhalt von Berichten oder Menüs, Verfügbarkeit von Funktionstasten) sowie Regeln für die Schnittstelle
- 2.1.3 Hardwareschnittstellen
- 2.1.4 Softwareschnittstellen
- 2.1.5 Kommunikationsschnittstellen (Netzwerkprotokolle etc.)
- 2.1.6 Speicherbeschränkungen
- 2.1.7 Betriebsoperationen (Operationsmodi, Dauer interaktiver und nichtinteraktiver Operationen, unterstützende Datenverarbeitungsfunktionen, Sicherungs- und Wiederherstellungsoperationen)
- 2.1.8 Möglichkeiten der lokalen Anpassung

2.1.7 **Betriebsoperationen**
Neben den eigentlichen Produktfunktionen bietet ein System meist noch eine Menge allgemeiner Operationen, die für den Betrieb der Software notwendig oder hilfreich sind. Dazu gehören beispielsweise unterstützende Datenverarbeitungsfunktionen (wie z.B. Konsistenzprüfungen oder allgemeine Statistiken über die Datenvolumen) sow Sicherungs- und Wiederherstellungsoperationen. Außerdem unterstützen viele Systeme verschiedene Operationsmodi, i denen unterschiedliche Produktfunktionen und Betriebsoperationen verfügbar beziehungsweise nicht verfügbar sind. Beispielsweise bietet eine Bank-Software einen Wartungsmodus, während dessen Kunden keine Transaktionen veranlassen können, weil Jahresabschlüsse berechnet werden.
In diesem Abschnitt werden diese allgemeinen Betriebsoperationen, die relevanten Operationsmodi und die Dauer und Häufigkeit interaktiver und nichtinteraktiver Operationen aufgeführt.

2.1.8 **Möglichkeiten der lokalen Anpassung**
Systeme bieten häufig die Möglichkeit, dass Benutzer lokale Anpassungen vornehmen können. Diese werden hier beschrieben.

Bsp.: *Der Ladenbesitzer soll beispielsweise einstellen können, wie viele Benutzer maximal gleichzeitig auf den Ladenrechner zugreifen dürfen.*

Die Norm ISO 9241-11:1998 (1998) weist Individualisierbarkeit als anzustrebendes Qualitätsmerkmal aus. Beispielswei sollen Schriftgrößen einstellbar sein. Möglichkeiten der lokalen Anpassung der Benutzungsschnittstelle werden jedoch i Abschnitt 2.3 beschrieben.

Inhalt der Anforderungsspezifikation nach IEEE Standard 830.1998

2.2 Produktfunktionen
- Zusammenfassung der Funktionen

2.3 Charakteristika der Benutzer
- Bildungsstand, Erfahrung, technische Kenntnisse

2.4 Einschränkungen
Beispiele:
- Gesetzliche Rahmenbedingungen
- Hardwarebeschränkungen
- parallelisierte Ausführung
- erforderliche Zuverlässigkeit
- sicherheitskritische Aspekte
- Datenschutzaspekte

2.5 Annahmen und Abhängigkeiten
- z.B. Betriebssystem ist auf Hardware verfügbar

Inhalt der Anforderungsspezifikation nach IEEE Standard 830.1998

2.2 Produktfunktionen
- Zusammenfassung der Funktionen

2.3 Charakteristika der Benutzer
- Bildungsstand, Erfahrung, technische Kenntnisse

2.4 Einschränkungen
Beispiele:
- Gesetzliche Rahmenbedingungen
- Hardwarebeschränkungen
- parallelisierte Ausführung
- erforderliche Zuverlässigkeit
- sicherheitskritische Aspekte
- Datenschutzaspekte

2.5 Annahmen und Abhängigkeiten
- z.B. Betriebssystem ist auf Hardware verfügbar

2.2 **Produktfunktionen**
Hier werden alle Produktfunktionen, die unterstützt werden sollen, kurz zusammengefasst. Ihre Detaillierung folgt in Kapitel 3.

2.3 **Charakteristika der Benutzer**
In diesem Abschnitt werden die Charakteristika der Benutzer beschrieben. Dies kann in Form der oben eingeführten Personas geschehen. Alle für uns als Entwickler relevanten Details sollten erwähnt werden. Dazu gehören meist der Bildungsstand, die Erfahrung und vorhandene technische Kenntnisse.

2.4 **Einschränkungen**
Alle maßgebenden Einschränkungen, die die Entwicklung betreffen, werden in diesem Abschnitt aufgeführt. Dazu gehören sowohl die so genannten nicht-funktionalen Produktattribute als auch organisatorische Rahmenbedingungen. Hier sind zum Beispiel gesetzliche Rahmenbedingungen, weitere Hardwarebeschränkungen, Zwang zur parallelisierten Ausführung, erforderliche Zuverlässigkeit, sicherheitskritische Aspekte sowie Aspekte des Datenschutzes zu nennen.

2.5 **Annahmen und Abhängigkeiten**
Unsere Entwicklung startet oft mit Annahmen, bei denen wir zum gegenwärtigen Zeitpunkt nicht sicher sind, ob sie zutreffen. Wenn die Entwicklung von solche Annahmen abhängt, stellen die Annahmen ein Risiko dar, das wir in diese Abschnitt explizit benennen. Der Architekt kann dann versuchen, die potentiellen Auswirkungen dieser Risiken im Architekturentwurf zu minimieren.

Jede Abhängigkeit von Dritten stellt ein weiteres Risiko dar. Sie führen zur Annahme, dass die, von denen wir abhänge die von uns erwünschte Leistung erbringen. Aus diesem Grund werden alle Abhängigkeiten hier auch explizit aufgeführ

Inhalt der Anforderungsspezifikation nach IEEE Standard 830.1998

3. Detaillierte Beschreibung

3.1 Externe Schnittstellen
- 3.1.1 Benutzungsschnittstelle
- 3.1.2 Hardwareschnittstelle
- 3.1.3 Softwareschnittstelle
- 3.1.4 Kommunikationsschnittstelle

3.2 Produktfunktionen

3.3 Performanzanforderungen

3.4 Entwurfseinschränkungen
z.B. Standards

3.5 Softwaresystemattribute
z.B. Zuverlässigkeit, Verfügbarkeit, Sicherheit, Wartbarkeit, Portabilität

3.6 Andere Anforderungen

Den größten Anteil der Anforderungsspezifikation nimmt das Kapitel 3 ein, in dem die Anforderungen in größerem Detail beschrieben werden.

3. Detaillierte Beschreibung

3.1 **Externe Schnittstellen**
In den vier folgenden Abschnitten werden die verschiedenen Schnittstellen im Detail beschrieben. Auf die Spezifikatior der Benutzungsschnittstelle gehen wir an dieser Stelle besonders ein.

3.1.1 **Benutzungsschnittstelle**
Die Benutzungsschnittstelle kann durch einen GUI-Prototypen beschrieben werden. Wenn dieser ausführbar is dann werden im Text hier die wesentlichen Interaktionskonzepte beschrieben; ansonsten wird auf den ausführbaren Prototypen verwiesen, der vom Kunden abgenommen sein muss. Ist der GUI-Prototyp nicht ausführbar, dann kann man hier die Screenshots und Skizzen einfügen, die in der Soll-Analyse erstellt wurden, um die Benutzungsschnittstelle vorzuführen. Es genügt jedoch nicht, nur Bildschirmmasken aufzulisten. Es mu auch der Zusammenhang der Dialoge und der anderen Interaktionselementen mit den Anwendungsfällen (d.h. der Funktionalität im Kontext des Arbeitsflusses, der unterstützt werden soll) beschrieben werden. Für jede Eingabemöglichkeit muss beschrieben sein, was damit ausgelöst werden kann und welche Ausgabe erfolgt. Ebenso muss die Navigation zwischen allen Dialogen und etwaige Zustände der Schnittstelle beschrieben sein. Die Beschreibung soll so genau sein, dass man daraus ohne weitere Schritte ein Benutzerhandbuch erstellen kann. Die Entscheidung, wie die Benutzungsschnittstelle auszusehen hat, liegt also nicht beim Programmierer später, sondern steht von Anfang an fest.

3.1.2 **Hardwareschnittstelle**

3.1.3 **Softwareschnittstelle**

3.1.4 **Kommunikationsschnittstelle**

3.2 **Produktfunktionen**
In diesem Abschnitt werden die Produktfunktionen im Detail beschrieben. Dieser Abschnitt bildet in der Regel mit der Benutzungsschnittstelle zusammen den umfangreichsten Teil. Deshalb ist eine sinnvolle Glieder unabdingbar. Wir werd weiter unten auf alternative Gliederungen eingehen. Der Zusammenhang zwischen Benutzungsschnittstellen (sowie alle anderen Schnittstellen) und der hier beschriebenen Produktfunktionen muss klar gemacht werden.

2008-11-04
Software-Projekt
└─Anforderungsanalyse
└─Anforderungsspezifikation
└─Inhalt der Anforderungsspezifikation nach IEEE Standard 830.1998
Inhalt der Anforderungsspezifikation nach IEEE Standard 830.1998
3. Detaillierte Beschreibung
3.1 Externe Schnittstellen
3.1.1 Benutzungsschnittstelle
3.1.2 Hardwareschnittstelle
3.1.3 Softwareschnittstelle
3.1.4 Kommunikationsschnittstelle
3.2 Produktfunktionen
3.3 Performanzanforderungen
3.4 Entwurfseinschränkungen
z.B. Standards
3.5 Softwaresystemattribute
z.B. Zuverlässigkeit, Verfügbarkeit, Sicherheit, Wartbarkeit, Portabilität
3.6 Andere Anforderungen

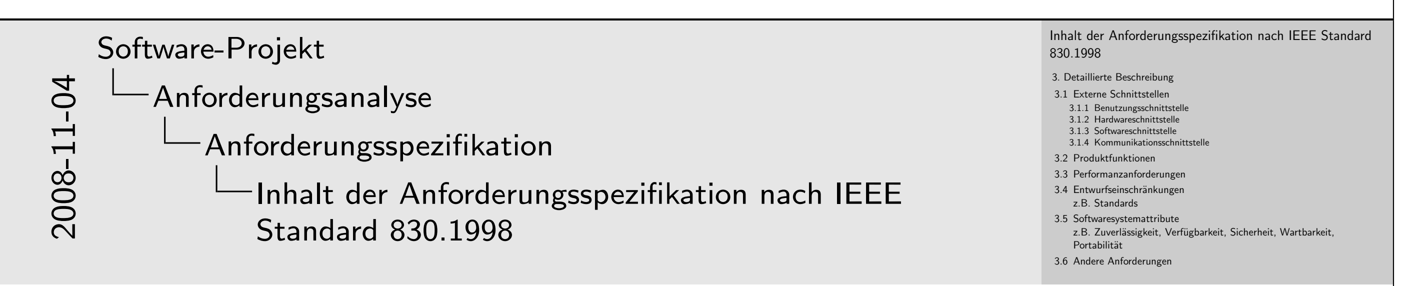
3.3 **Performanzanforderungen**
Funktionale Anforderungen beschreiben die erwartete Ausgabe auf eine Eingabe hin. Bei Echtzeitsystemen mit harten Deadlines muss nicht nur die erwartete Ausgabe angegeben werden, sondern auch nach welcher Zeit die Antwort erfolgen muss, da sie ansonsten nicht mehr gebraucht wird. Bei Systemen ohne Echtzeitanforderungen hingegen werde zeitliche Aspekte häufig nicht explizit angegeben. Das ist jedoch falsch. Denn auch bei interaktiven Systemen erwarter wir eine zeitnahe Reaktion des Systems auf unsere Eingabe hin, weil wir das Programm sonst nicht verwenden wollen. Erfolgen keine konkreten Angaben zur maximalen Reaktionszeit ist es schwer, über die Vertragserfüllung bei der Abga zu diskutieren. Deshalb sollte man grundsätzlich Aussagen zur erforderlichen Performanz jeder Funktionalität machen. Dies kann relativ zum Datenvolumen erfolgen oder in absoluten Zeitangaben. Im letzteren Fall nimmt das Mengengerü eine noch gewichtigere Stellung ein. Denn nur durch das Mengengerüst können wir zu absoluten Zeiten kommen, die später überprüfbar sind.

In diesem Abschnitt ist der Platz für die Beschreibung der Performanzanforderungen aller Produktfunktionen. Alternat kann man diese Anforderungen auch bei der Beschreibung der Produktfunktionen angeben und dann auf diesen Abschnitt verzichten. Das Prinzip der Trennung von Aspekten, dem wir bei einer Anforderungsspezifikation folgen, leg aber einen eigenen Abschnitt nahe. Die Beschreibung kann z.B. durch einfache Tabellen erfolgen, in denen die Produktfunktionen aufgeführt sind und die maximale Reaktionszeit in Abhängigkeit der Eingabe angegeben wird.

3.4 **Entwurfseinschränkungen**
Implementierungsdetails gehören – wie schon mehrfach gesagt – nicht in die Anforderungsspezifikation. Es gibt aber dennoch häufig Aspekte, die beim Entwurf berücksichtigt werden müssen. Dazu gehören z.B. Standards, an die sich d Software halten muss. In der Luft- und Raumfahrt beispielsweise gibt es Codierungsstandards wie MISRA, deren Einhaltung zuverlässigere Systeme verspricht. Hersteller von Software müssen sich an solche Standards halten, sonst wird ihr System nicht verwendet. In diesem Abschnitt werden aus diesem Grund alle weiteren Einschränkungen des Entwurfs beschrieben, sofern sie aus Kundensicht relevant sind.

3.5 **Softwaresystemattribute**
Neben Performanzanforderungen gibt es noch weitere Systemattribute, zu denen die Anforderungsspezifikation Aussag machen muss. Dazu gehören Zuverlässigkeit, Verfügbarkeit, Sicherheit, Wartbarkeit, Portabilität und weitere Softwarequalitäten (oft auch nichtfunktionale Anforderungen genannt). Wir werden weiter unten noch darauf zu sprechen kommen, dass diese Art Anforderungen konkret beschrieben werden müssen. Die Forderung „Das System mu sicher sein" ist viel zu allgemein als dass man sie erfüllen könnte.

3.6 **Andere Anforderungen**
In diesem Abschnitt können weitere relevante Anforderungen beschrieben werden, die in keine der oben genannten Abschnitte passen.



2008-11-04

Software-Projekt
└─Anforderungsanalyse
└─Anforderungsspezifikation
└─Inhalt der Anforderungsspezifikation nach IEEE Standard 830.1998

Inhalt der Anforderungsspezifikation nach IEEE Standard 830.1998

3. Detaillierte Beschreibung
3.1 Externe Schnittstellen
3.1.1 Benutzungsschnittstelle
3.1.2 Hardwareschnittstelle
3.1.3 Softwareschnittstelle
3.1.4 Kommunikationsschnittstelle
3.2 Produktfunktionen
3.3 Performanzanforderungen
3.4 Entwurfseinschränkungen
z.B. Standards
3.5 Softwaresystemattribute
z.B. Zuverlässigkeit, Verfügbarkeit, Sicherheit, Wartbarkeit, Portabilität
3.6 Andere Anforderungen

Ergänzungen zum Standard:

Die Anforderungsspezifikation nach IEEE Standard 830.1998 verlangt nirgendwo, dass die Ergebnisse der Ist-Analyse beschrieben werden sollen. Diese Ergebnisse sind jedoch wichtig, um die Anforderungen zu verstehen und einschätzen zu können. Man sollte deshalb entweder ein eigenes Dokument hierfür anlegen, das in der Anforderungsspezifikation referenziert wird, oder aber im Einführungskapitel einen entsprechenden Abschnitt verfassen.

Außerdem weist der Standard an keiner Stelle darauf hin, dass auch die möglichen zukünftigen Änderungen der Anforderungen beschrieben sein sollten. Diese sind jedoch wichtig, damit der Architekt die Architektur so entwerfen kann, dass diese Änderungen – so sie eintreffen – relativ einfach umgesetzt werden können. Aus diesem Grunde empfiehlt sich in Kapitel 2 beziehungsweise Kapitel 3 (je nach Detaillierungsgrad) ein Abschnitt zu zukünftigen Entwicklungen der Anforderungen.

Im Datenmodell modellieren wir die Objekte der Anwendungsdomäne, die für die Beschreibung unserer Anforderungen relevant sind. Im Standard wird kein Ort explizit genannt, an dem das Datenmodell beschrieben werden soll. Hier bietet sich ein Unterabschnitt in Abschnitt 2.2 an.

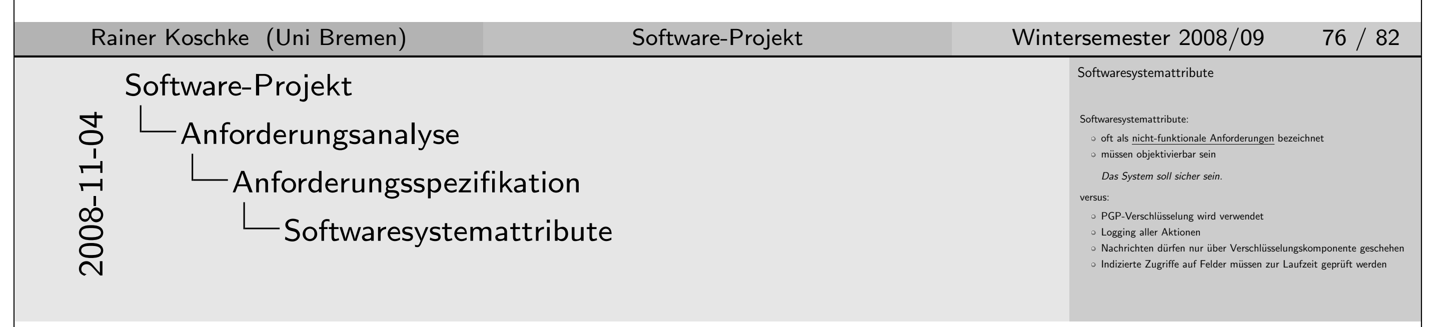
Softwaresystemattribute:

- oft als nicht-funktionale Anforderungen bezeichnet
- müssen objektivierbar sein

  Das System soll sicher sein.

versus:

- PGP-Verschlüsselung wird verwendet
- Logging aller Aktionen
- Nachrichten dürfen nur über Verschlüsselungskomponente geschehen
- Indizierte Zugriffe auf Felder müssen zur Laufzeit geprüft werden



Wie oben beim Abschnitt 3.5 *Softwaresystemattribute* erwähnt sind neben den funktionalen Anforderungen auch die so genannten nicht-funktionalen Anforderungen, wie Portierbarkeit etc., zu beschreiben. Wie wir auch schon oben diskutiert haben, sollen alle Anforderungen überprüfbar sein. Das heißt die Aussagen zu den nicht-funktionalen Eigenschaften müssen so genau und konkret sein, dass man eine Prüfprozedur angeben kann, um bei Auslieferung überprüfen zu können, ob die Anforderung erfüllt wurde. Die Forderung „Das System muss sicher sein" genügt diesem Anspruch nicht, weil nicht klar ist, wogegen das System gesichert werden soll. Konkreter sollte man also etwa fordern:

- Alle Netzwerktransaktionen werden mit PGP verschlüsselt.
- Alle Operationen des Administrators werden aufgezeichnet.
- Alle Benutzer müssen sich mit einem mindestens acht Zeichen langen Passwort authentifizieren.

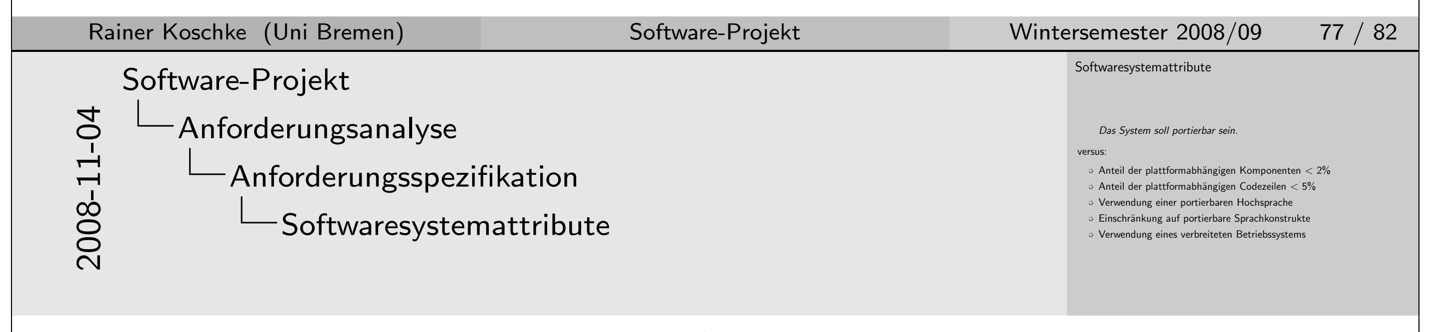
Diese Aussagen sind überprüfbar. Interessanterweise stellen wir an dieser Stelle fest, dass aus den so genannten nicht-funktionalen Eigenschaften durch die Konkretisierung funktionale Eigenschaften werden können. Die Trennung in funktionale und nicht-funktionale Eigenschaften ist also etwas künstlich bei allen Eigenschaften, die externe Eigenschaften betreffen. Externe Eigenschaften sind solche, die der Benutzer direkt wahrnehmen kann. Interne Eigenschaften betreffen vielmehr den inneren Aufbau.

Softwaresystemattribute

Das System soll portierbar sein.

versus:

- Anteil der plattformabhängigen Komponenten $< 2\%$
- Anteil der plattformabhängigen Codezeilen $< 5\%$
- Verwendung einer portierbaren Hochsprache
- Einschränkung auf portierbare Sprachkonstrukte
- Verwendung eines verbreiteten Betriebssystems



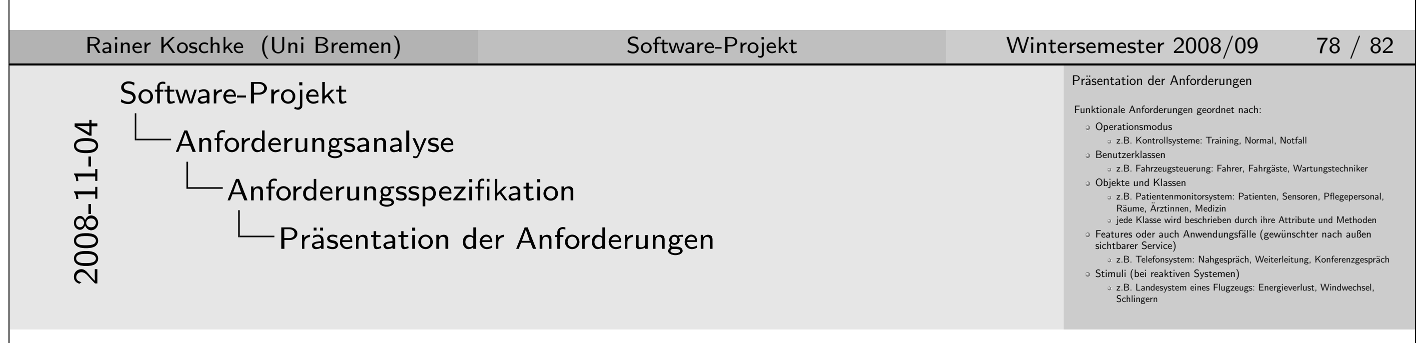
Aber auch Anforderungen zu inneren Eigenschaften der Software müssen hinreichend genau angegeben sein. Die Aussage „Das System soll portierbar sein“, die eine innere Qualität darstellt, kann nicht überprüft werden, weil nicht klar ist, wohin portiert werden soll und was Portierbarkeit genau bedeuten soll. Besser muss angegeben werden, auf welcher Hardware und für welches Betriebssystem die Software portiert werden soll. Dann muss der Begriff *portierbar* konkretisiert werden, z.B. auf die folgende Weise:

- Der Anteil der plattformabhängigen Komponenten soll weniger als 2% ausmachen.
- Der Anteil der plattformabhängigen Codezeilen soll weniger als 5% sein.
- Es soll eine Hochsprache als Implementierungssprache verwendet werden, die auf allen Plattformen verfügbar ist, auf d portiert werden soll.
- Alle verwendeten Konstrukte der Programmiersprache müssen auf allen Zielplattformen verfügbar sein.

Präsentation der Anforderungen

Funktionale Anforderungen geordnet nach:

- Operationsmodus
  - z.B. Kontrollsysteme: Training, Normal, Notfall
- Benutzerklassen
  - z.B. Fahrzeugsteuerung: Fahrer, Fahrgäste, Wartungstechniker
- Objekte und Klassen
  - z.B. Patientenmonitorsystem: Patienten, Sensoren, Pflegepersonal, Räume, Ärztinnen, Medizin
  - jede Klasse wird beschrieben durch ihre Attribute und Methoden
- Features oder auch Anwendungsfälle (gewünschter nach außen sichtbarer Service)
  - z.B. Telefonsystem: Nahgespräch, Weiterleitung, Konferenzgespräch
- Stimuli (bei reaktiven Systemen)
  - z.B. Landesystem eines Flugzeugs: Energieverlust, Windwechsel, Schlingern



Wie oben erwähnt, kommt der Strukturierung des Abschnitts 3.2 *Produktfunktionen* eine große Bedeutung zu. Dieses Kapitel ist sehr umfangreich und alle Leser sollen sich möglichst einfach darin zurecht finden.

Welche Kategorisierung zu verwenden ist, hängt von der Art der Anwendung ab. Bei reaktiven Systemen, also z.B. Steuerungssystemen, kann man nach den Stimuli gruppieren, auf die die Software reagieren muss. Bei einer Software, die das Landen eines Flugzeugs steuert, könnte man z.B. nach plötzlichem Energieverlust, Windwechsel oder Schlingern etc. ordnen. Zu den allgemeineren Ordnungskriterien, die bei sehr vielen Arten von Systemen Anwendung finden können, zählen Operationsmodi. Kontrollsysteme haben z.B. häufig die Operationsmodi Training, Normal und Notfall. Ein weiteres allgemeines Kriterium ist das der Benutzerklassen. Die Software eines Busses könnte unterscheiden in Produktfunktionen, die Fahrer, Fahrgäste oder Wartungstechniker betreffen. Bei objektorientiertem Vorgehen ergeben sich in der Analyse Objekte und Klassen, nach denen man gruppieren könnte. In einem Patientenmonitorsystem erwarten wir als Klassen z.B. Patienten, Sensoren, Pflegepersonal, Räume, Ärztinnen, Medizin. Jede dieser Klassen wird dann beschrieben durch ihre Attribute und Nachrichten, die sie versteht. Schließlich können wir auch nach Anwendungsfällen kategorisieren. Bei einem Telefonsystem wären das zum Beispiel das Nahgespräch, die Weiterleitung und das Konferenzgespräch.

Wie immer man sich entscheidet, man sollte in jedem Falle die Kategorisierung uniform anwenden. Es kann dennoch sinnvoll sein, Kategorien zu kombinieren, was kein Widerspruch zur Uniformität sein muss. Dem Ziel der Uniformität genügend könnte man z.B. auf erster Ebene nach Benutzerklassen ordnen und auf zweiter Ebene nach Anwendungsfällen.

Wiederholungsfragen I

- Was ist an der Erhebung der Anforderungen so schwierig?
- Erläutern Sie das Kano-Modell. Was folgt daraus für die Anforderungsspezifikation?
- Wozu dient die Anforderungsspezifikation und welche Folgen haben ihre Mängel?
- Was sind die Schritte der Anforderungsspezifikation?
- Was sind Personas und welches Ziel wird mit ihnen verfolgt?
- In welchen Phasen des Entwicklungsprozesses können Personas verwendet werden?
- Was ist das Ziel der Ist-Analyse und aus welchen Schritten besteht sie?
- Erläutern Sie Techniken zur Ist-Analyse. Bewerten Sie sie.
- Insbesondere: Auf welchen Prinzipien basiert das Interview im Kontext und wie wird es durchgeführt?

Wiederholungsfragen II

- Welche Arten des Prototypings gibt es und wozu dienen sie?
  - Wegwerf-, technischer und evolutionärer Prototyp
  - horizontaler versus vertikaler Prototyp
  - passive, aktive, interaktive Storyboards
- Welche Eigenschaften der Anforderungsspezifikation streben wir an?
- Welche Bestandteile hat eine Anforderungsspezifikation?
- Welche Systemattribute spielen in der Anforderungsspezifikation eine Rolle und worauf ist bei ihrer Spezifikation zu achten?

1 **EN ISO 9241-10:1995 1995** : Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten – Teil 10: Grundsätze der Dialoggestaltung. Europäische Norm, ISO-Standard. 1995

2 **IEEE Standard 830.1998 1998** : IEEE Recommended Practice for Software Requirements Specifications, IEEE Standard 830.1998. 1998

3 **IEEE Std 610.12-1990 1990** : IEEE Standard Glossary of Software Engineering Terminology. IEEE Standard 610.12-1990. 1990

4 **ISO 9241-11:1998 1998** : Ergonomic requirements for office work with visual display terminals (VDTs) – Part 11: Guidance on usability. ISO-Standard. 1998

5 **ISO/IEC-Standard 25000 2005** ISO/IEC 25000:2005 Software Engineering – Software product Quality Requirements and Evaluation (SQuaRE). 2005. – umgesetzt in DIN 66272

6 **ISO/IEC-Standard 9126 2001** ISO/IEC 9126-1:2001 Software engineering – Product quality. 2001. – umgesetzt in DIN 66272

7 **Kano 1984** KANO, N.: Attractive Quality and Must-be Quality. In: Journal of the Japanese Society for Quality Control 4 (1984), S. 39–48

8 **Ludewig 2003** LUDEWIG, Jochen: Einführung in die Softwaretechnik. Vorlesungs-Skriptum. 2003

9 **Pressman 2003** PRESSMAN, Roger: Software Engineering – A Practioner's Approach. Fünfte Ausgabe. McGraw-Hill, 2003

0 **Sommerville 2004** SOMMERVILLE, Ian: Software Engineering. Siebte Ausgabe. Addison-Wesley, 2004